2nd Symposium on Advances in Approximate Bayesian Inference, 2019 1–11

Efficient Bayesian Inference for Nested Simulators

Bradley Gram-Hansen BRADLEY@ROBOTS.OX.AC.UK
Christian Schroeder de Witt CS@ROBOTS.OX.AC.UK
Robert Zinkov ZINKOV@ROBOTS.OX.AC.UK
University of Oxford

Saeid Naderiparizi SAEIDNP@CS.UBC.CA
Adam Scibior ADAM.SCIBIOR@INVERTED.AI
Andreas Munk ANDREAS.M.M64@GMAIL.COM
Mehrdad Ghadiri MEHRDAD29@GMAIL.COM
Frank Wood FWOOD@CS.UBC.CA
University of British Columbia

Philip Torr PHILIP.TORR@ENG.OX.AC.UK
Yee Whye Teh Y.W.TEH@STATS.OX.AC.UK
Atilim Gunes Baydin GUNES@ROBOTS.OX.AC.UK
Tom Rainforth RAINFORTH@STATS.OX.AC.UK
University of Oxford

Abstract

We introduce two approaches for conducting efficient Bayesian inference in stochastic simulators containing nested stochastic sub-procedures, i.e., internal procedures for which the density cannot be calculated directly such as rejection sampling loops. The resulting class of simulators are used extensively throughout the sciences and can be interpreted as probabilistic generative models. However, drawing inferences from them poses a substantial challenge due to the inability to evaluate even their unnormalised density, preventing the use of many standard inference procedures like Markov Chain Monte Carlo (MCMC). To address this, we introduce inference algorithms based on a two-step approach that first approximates the conditional densities of the individual sub-procedures, before using these approximations to run MCMC methods on the full program. Because the sub-procedures can be dealt with separately and are lower-dimensional than that of the overall problem, this two-step process allows them to be isolated and thus be tractably dealt with, without placing restrictions on the overall dimensionality of the problem. We demonstrate the utility of our approach on a simple, artificially constructed simulator.

1. Introduction and Background

Stochastic simulators are used in a myriad of scientific and industrial settings, such as epidemiology (Patlolla et al., 2004), physics (Heermann, 1990), engineering (Hangos and Cameron, 2001) and climate modelling (Held, 2005). They can be complex and high-dimensional, often incorporating domain-specific expertise accumulated over many years of research and development.

As shown by the probabilistic programming (Gordon et al., 2014; van de Meent et al., 2018; Baydin et al., 2019) and approximate Bayesian computation (ABC) (Csilléry et al., 2010; Marin et al., 2012) literatures, these simulators can be interpreted as probabilistic generative models, implicitly defining a probability distribution over their internal variables

and outputs. As such, they form valid targets for drawing Bayesian inferences. In particular, by constraining selected internal variables or outputs to take on specific values, we implicitly define a conditional distribution, or posterior, over the remaining variables. This effectively allows us, amongst other things, to run the simulator in "reverse", fixing the outputs to some observed values and figuring out what parameter values might have led to them. For example, given a simulator for visual scenes, we can run inference on the simulator with an observed image to predict what objects are present in the scene (Kulkarni et al., 2015).

Though recent advances in probabilistic programming systems (PPSs, Tran et al. (2017); Bingham et al. (2019); Baydin et al. (2019); Casado et al. (2017)) have provided convenient mechanisms for encoding, reasoning about, and constructing inference algorithms for such simulators, performing the necessary inference is still often extremely challenging, particularly for complex or high-dimensional problems.

In this paper, we consider a scenario where this inference is particularly challenging to perform: when the simulator makes calls to nested stochastic sub-procedures (NSSPs). These NSSPs can take several different forms, such as internal rejection sampling loops, separate inference procedures, external sub-simulators we have no control over, or even real-world experiments. Their unifying common feature is that the density of their outputs cannot be evaluated up to an input-independent normalising constant in closed form. This, in turn, means the normalised density of the overall simulator cannot be evaluated, preventing one from using most common inference methods, including almost all Markov chain Monte Carlo (MCMC) and variational methods. Though some inference methods can still be applied in these scenarios, such as nested importance sampling (Rainforth, 2018), these tend to scale very poorly in the dimensionality and often even have fundamentally slower convergence rates than standard Monte Carlo approaches (Rainforth et al., 2018).

To address this issue, we introduce two new approaches for performing inference in such models. Both are based around approximating the individual NSSPs. The first approach directly approximates the conditional density of the NSSP outputs using an amortized inference artefact. This then forms a surrogate density for the NSSP, which, once trained, is used to replace it.

While this first approach is generally applicable, our second approach focuses on the specific case where the unnormalized density of the NSSP can be evaluated in isolation (such as a nested probabilistic program or rejection sampling loop), but its normalizing constant depends on the NSSP inputs. Here, we train a regressor to approximate the normalising constant of the NSSP as a function of its inputs. Once learnt, this allows the NSSP to be collapsed into the outer program: the ratio of the known unnormalised density and the approximated normalizing constant can be directly used as a factor in the overall density.

Both approaches lead to an approximate version of the overall unnormalised density, which can then be used as a target for conventional inference methods like MCMC and variational inference. Because these approximations can be calculated separately for each NSSP, this allows them to scale to higher dimensional overall simulators far more gracefully than existing approaches, opening the door to tractably running inference for more complex problems. Furthermore, once trained, the approximations can be reused for different datasets and configurations of the outer simulator, thereby helping amortise the cost of running multiple different inferences for no extra cost. The approaches themselves are also amenable to automation, making them suitable candidates for PPS inference engines.

2. Approximating Sub-Procedures

We now introduce our two approaches for approximating NSSPs and show how these, in turn, produce efficient inference algorithms for the overall simulator. Both our approaches involve the gradient-based learning of a neural-network-based amortised approximation for each NSSP that takes in the NSSP inputs and either returns and approximation of the density of the outputs (method 1) or the normalizing constant (method 2).

For any simulator or *program*, we can define the program density over valid program traces $x_{1:n_x}$ as (Rainforth, 2017, Section 4.3.2):

$$p(x_{1:n_x}) \propto \gamma(x_{1:n_x}) = \prod_{j=1}^{n_x} f_{a_j}(x_j|\phi_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) \tag{1}$$

where n_x is the length of the trace; each $f_{a_j}(x_j|\phi_j)$ represents the density of the j^{th} random draw, which is made at location a_j and takes in parameters ϕ_j; and n_y is a number of "observations", each of which factor the trace density by $g_{b_k}(y_k|\psi_k)$, where b_k is the location of this observation statement, y_k is the observed value, and ψ_k are parameters of the factorization. Here all terms—i.e., x_j, n_x, a_j, ϕ_j, n_y, b_k, y_k, and ψ_k—may be random variables, but each is deterministically calculable from the trace $x_{1:n_x}$ (see Rainforth (2017, Section 4.3.2))

A NSSP can now be formally defined as a $f_{a_j}(x_j|\phi_j)$ term which cannot be directly evaluated exactly, but where for a given ϕ_j either **[Case A]** we can draw samples from $f_{a_j}(x_j|\phi_j)$ directly and/or **[Case B]** $f_{a_j}(x_j|\phi_j)$ corresponds to the normalized density of a nested probabilistic program that we can draw approximate samples from by running an separate inference procedure. Many simulators contain such sampling procedures (Di Pasquale et al., 2015; Gleisberg et al., 2009; Smith et al., 2006; Heermann, 1990; Rainforth, 2018; Baydin et al., 2019), and it is these simulators that we target with our inference schemes.

We can denote the unnormalized density for a program containing NSSPs as

$$\gamma(x_{1:n_x}) = P_{pr}(x_{1:n_x}) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) \tag{2}$$

$$\text{where} \quad P_{pr}(x_{1:n_x}) := \prod_{\{j\in 1:n_x | a_j \notin S_r\}} f_{a_j}(x_j|\phi_j) \prod_{\{j\in 1:n_x | a_j \in S_r\}} P^{in}_{a_j}(x_j|\phi_j) \tag{3}$$

is a representation of the "forward" or "prior" program which ignores all conditioning statements; $S_r = \{a_1, \ldots, a_n\}$ represents the set of addresses that produce intractable densities; and we use $P^{in}_{a_j}(x_j|\phi_j)$ to distinguish the NSSPs from tractable sampling terms. Both our methods are now based on replacing each of the $P^{in}_{a_j}(x_j|\phi_j)$ with an approximation, for which we only need to consider the prior program. Once learned, these can then be used to construct a directly evaluable approximate target density $\hat{\gamma}(x_{1:n_x})$ by replacing each $P^{in}_{a_j}(x_j|\phi_j)$ in (3), then running an MCMC sampler on $\hat{\gamma}(x_{1:n_x})$.

2.1. Method 1: Surrogate Replacement

Our first method replaces each $P^{in}_{a_j}(x_j|\phi_j)$ by an approximate surrogate $q^{in}_{a_j}(x_j|\phi_j;\eta_{a_j})$:

$$P_{pr}(x_{1:n_x}) \simeq q(x_{1:n_x};\kappa) := \prod_{\{j\in 1:n_x | a_j \notin S_r\}} f_{a_j}(x_j|\phi_j) \prod_{\{j\in 1:n_x | a_j \in S_r\}} q^{in}_{a_j}(x_j|\phi_j;\eta_{a_j}) \tag{4}$$

where $\kappa = \{\eta_{a_j}; a_j \in S_r\}$ are the surrogate parameters. As per existing amortized variational approaches (Kingma and Welling, 2014; Rezende and Mohamed, 2015; Le et al., 2016; Ritchie et al., 2016; Paige and Wood, 2016), each $q^{in}_{a_j}(x_j|\phi_j;\eta_{a_j})$ is taken as a variational distribution parametrized by deep neural network with weights η_{a_j} and which takes ϕ_j as its input.

Training of these networks is done by minimising the Kullback–Leibler (KL) divergence from $P_{pr}(x_{1:n_x})$ to $q(x_{1:n_x};\kappa)$ (Paige and Wood, 2016)

$$\kappa^* = \underset{\kappa}{\operatorname{argmin}}\, \mathrm{KL}\left(P_{pr}||q_\kappa\right) = \underset{\{\eta_{a_j};a_j\in S_r\}}{\operatorname{argmin}}\, \mathbb{E}_{P_{pr}}\left[-\sum_{\{j\in 1:n_x|a_j\in S_r\}} \log q^{in}_{a_j}(x_j|\phi_j;\eta_{a_j})\right]. \tag{5}$$

This minimization can be done using stochastic gradient descent where the updates for NSSP $r \in S_r$ use the following gradient estimate (see Appendix A)

$$\nabla_{\eta_r}\mathrm{KL} \approx -\frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{n_x^n}\mathbb{I}(r=a_j^n)\nabla_{\eta_r}\log(q_r^{in}(x_j^n|\phi_j^n;\eta_r)), \quad \text{where} \quad x^n_{1:n_x} \overset{\text{i.i.d.}}{\sim} P_{pr}(x_{1:n_x}) \tag{6}$$

and the $x^n_{1:n_x}$ can be shared such that the variational approximations for each $r \in S_r$ are made simultaneously.

Carrying out these updates requires us to draw samples from P_{pr}. If all of our NSSPs satisfy [Case A], this is not a problem as by assumption we can then draw samples from each $P^{in}_{a_j}(x_j|\phi_j)$ and, in turn, samples from P_{pr}. However, if our program contains NSSPs which only satisfy [Case B], this will require us to run a separate nested inference (Rainforth, 2018) to generate the required x_j^n from the corresponding ϕ_j. Though this may be potentially non-trivial, it is, crucially, far easier that running inference on the overall program: because P_{pr} itself does not include any conditioning statements, generating these samples does not require inference to be run for the outer program. As such, each nested inference problem constitutes its own isolated problem which is far simpler than the overall inference problem. In other words, the role of sampling from P_{pr} is only to generate example input-output pairs for each NSSP, with each surrogate than separately trained based on its local pairs.

2.2. Method 2: Normalisation Constant Approximation

If all of our NSSPs satisfy [Case B], this implies that each has a known unnormalised density on its internal variables and unknown input-dependent normalizing constant that causes a double-intractability. If the functional form for all these normalizing constant where known, this would be sufficient to collapse all the NSSPs into the outer program and produce a directly evaluable density for the overall program. Our second method thus looks to learn regressors to predict the normalizing constants and thereby facilitate this.

To formalize this, let us for now assume that the x_j returned by each NSSP corresponds to its full set of internal random draws $z^j_{1:n_x^j}$, i.e., $x_j = z^j_{1:n_x^j}$, such that we can write

$$P^{in}_{a_j}\left(x_j|\phi_j\right) = \gamma^{in}_{a_j}\left(x_j|\phi_j\right) \Big/ I^{in}_{a_j}(\phi_j) = \gamma^{in}_{a_j}\left(z^j_{1:n_x^j}\middle|\phi_j\right) \Big/ I^{in}_{a_j}(\phi_j) \tag{7}$$

where $\gamma^{in}_{a_j}(z^j_{1:n_x^j}|\phi_j)$ can be evaluated directly (because it is itself an unnormalized probabilistic program density of the form (1)), but $I^{in}_{a_j}(\phi_j)$ is an intractable normalization constant. If we now introduce a set of regressors $R_r(\phi_j;\tau_r)$, $\forall r \in S_r$ (with parameters τ_r) to approximate

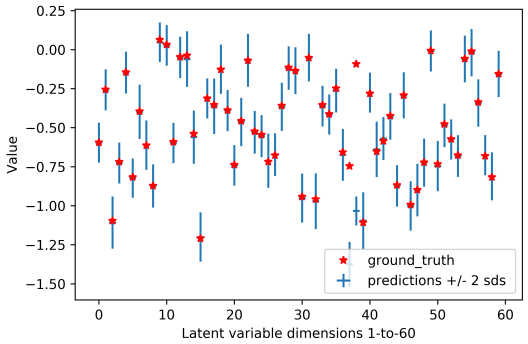

Figure 1: Posterior inferences of the means for each latent dimension from running method 1 with Hamiltonian Monte Carlo on nested Gaussian example. Each prediction comprises of the means and standard deviations for 5 independent runs of 10,000 samples. The red stars represent the ground true means for each dimension. More detailed results, shown in Figure 2, emphasize that our inference algorithm has encapsulated the marginal posteriors almost exactly.

each $I_r^{in}(\phi_j)$, we can approximate P_{pr} as

$$P_{pr}(x_{1:n_x}) \simeq \prod_{\{j\in 1:n_x | a_j \notin S_r\}} f_{a_j}(x_j|\phi_j) \prod_{\{j\in 1:n_x | a_j \in S_r\}} \frac{\gamma_{a_j}^{in}\left(z_{1:n_x^j}^j \middle| \phi_j\right)}{R_{a_j}(\phi_j;\tau_{a_j})}. \tag{8}$$

We can extend this approach to the case where $x_j = z_{1:n_x^j}^j$ by instead defining our reference measure in the space of $\mathcal{X}_a := \{x_j\}_{j\in 1:n_x | a_j \notin S_r} \cup \{z_{1:n_x^j}^j\}_{j\in 1:n_x | a_j \notin S_r}$ and using the pre-image of the prior program density: $P_{pr}\left(\mathcal{X}_a\right)$. We can then run inference in this pre-image space and rely on the law of the unconscious statistician to ensure the samples produces are from the desired posterior (see e.g. Rainforth (2017, Section 4.3.2)).

Learning the regressors $R_r(\phi_j;\tau_r)$ is done in an analogous manner to method one. Namely we run the program forward to gather pairs $\{\phi_j, \hat{I}_r(\phi_j)\}$ for each NSSP, where $\hat{I}_r(\phi_j)$ is an unbiased approximation of $I_r(\phi_j)$, and then use this as a training dataset for learning the regressor. Specifically, for each NSSP we train a neural network regressor to minimize the expected squared error between $R_r(\phi_j;\tau_r)$ and $\hat{I}_r(\phi_j)$. As shown in Appendix B, with a sufficiently expressive neural network, this scheme ensures $R_r(\phi_j;\tau_r) \to I_r(\phi_j)\,\forall\phi_j$ as the number of training pairs tends to infinity.

3. Experiments

In this section, we use a 60-d nested Guassian example, details of which are given in Appendix C. The model has been contrived so that we can analytically calculate the posterior means and therefore validate against ground truth values. Figure 2 demonstrates this for Method 1 (results for Method 2 are still being developed). We see that accurate inference was achieved for all but two of the marginal distributions (these were caused by issues in the stability of the neural network training, which is currently being investigated). Though still preliminary, these results are very promising in that they demonstrate that we are able to perform effective inference in far higher dimensions that can be realistically achieved by importance sampling based approaches, which are the current standard in the field.

References

Atılım Güneş Baydin, Lukas Heinrich, Wahid Bhimji, Lei Shao, Saeid Naderiparizi, Andreas Munk, Jialin Liu, Bradley Gram-Hansen, Gilles Louppe, Lawrence Meadows, Philip Torr, Victor Lee, Prabhat, Kyle Cranmer, and Frank Wood. Efficient probabilistic inference in the quest for physics beyond the standard model. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2019.

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

Mario Lezcano Casado, Atilim Gunes Baydin, David Martínez Rubio, Tuan Anh Le, Frank Wood, Lukas Heinrich, Gilles Louppe, Kyle Cranmer, Karen Ng, Wahid Bhimji, et al. Improvements to inference compilation for probabilistic programming in large-scale scientific simulators. *arXiv preprint arXiv:1712.07901*, 2017.

George Casella and Roger L. Berger. *Statistical Inference*. Thomson Learning. ISBN 978-0-534-24312-8. Google-Books-ID: 0x_vAAAAMAAJ.

Katalin Csilléry, Michael GB Blum, Oscar E Gaggiotti, and Olivier François. Approximate bayesian computation (abc) in practice. *Trends in ecology & evolution*, 25(7):410–418, 2010.

Valentina Di Pasquale, Salvatore Miranda, Raffaele Iannone, and Stefano Riemma. A simulator for human error probability analysis. *Reliability Engineering & System Safety*, 139: 17–32, 2015.

Tanju Gleisberg, Stefan Höche, F Krauss, M Schönherr, S Schumann, F Siegert, and J Winter. Event generation with sherpa 1.1. *Journal of High Energy Physics*, 2009(02):007, 2009.

Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, pages 167–181. ACM, 2014.

Bradley Gram-Hansen, Christian Schröder de Witt, Tom Rainforth, Philip HS Torr, Yee Whye Teh, and Atılım Güneş Baydin. Hijacking malaria simulators with probabilistic programming. *ICML Workshop on AI for Social Good*, 2019.

Katalin M Hangos and Ian T Cameron. *Process modelling and model analysis*, volume 4. Academic press London, 2001.

Dieter W Heermann. Computer-simulation methods. In *Computer Simulation Methods in Theoretical Physics*, pages 8–12. Springer, 1990.

Isaac M Held. The gap between simulation and understanding in climate modeling. *Bulletin of the American Meteorological Society*, 86(11):1609–1614, 2005.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4390–4399, 2015.

Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. *arXiv preprint arXiv:1610.09900*, 2016.

Jean-Michel Marin, Pierre Pudlo, Christian P Robert, and Robin J Ryder. Approximate bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

Brooks Paige and Frank Wood. Inference networks for sequential monte carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049, 2016.

Padmavathi Patlolla, Vandana Gunupudi, Armin R Mikler, and Roy T Jacob. Agent-based simulation tools in computational epidemiology. In *International Workshop on Innovative Internet Community Systems*, pages 212–223. Springer, 2004.

Kaare Brandt Petersen et al. The matrix cookbook.

Thomas William Gamlen Rainforth. *Automating inference, learning, and design using probabilistic programming*. PhD thesis, University of Oxford, 2017.

Tom Rainforth. Nesting probabilistic programs. *arXiv preprint arXiv:1803.06328*, 2018.

Tom Rainforth, Robert Cornish, Hongseok Yang, and Andrew Warrington. On nesting monte carlo estimators. In *International Conference on Machine Learning*, pages 4264–4273, 2018.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

T Smith, N Maire, A Ross, M Penny, N Chitnis, A Schapira, A Studer, B Genton, C Lengeler, Fabrizio Tediosi, et al. Towards a comprehensive simulation model of malaria epidemiology and control. *Parasitology*, 135(13):1507–1516, 2008.

Thomas Smith, Nicolas Maire, Klaus Dietz, Gerry F Killeen, Penelope Vounatsou, Louis Molineaux, and Marcel Tanner. Relationship between the entomologic inoculation rate and the force of infection for plasmodium falciparum malaria. *The American journal of tropical medicine and hygiene*, 75(2_suppl):11–18, 2006.

Dustin Tran, Matthew D Hoffman, Rif A Saurous, Eugene Brevdo, Kevin Murphy, and David M Blei. Deep probabilistic programming. *arXiv preprint arXiv:1701.03757*, 2017.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

Appendix A. Derivation of Gradient of The Variational Objective

For a given simulator, or *program*, we denote the proposal for the program as:

$$P_{pr}(x_{1:n_x}) \simeq q(x_{1:n_x};\kappa) := \prod_{\{j\in 1:n_x | a_j \notin S_r\}} f_{a_j}(x_j|\phi_j) \prod_{\{j\in 1:n_x | a_j \in S_r\}} q^{in}_{a_j}(x_j|\phi_j;\eta_{a_j}) \tag{9}$$

where $\kappa = \{\eta_{a_j}; a_j \in S_r\}$ are the variational parameters. Using the information projection we construct the variational objective as follows:

$$\begin{aligned}
J(\kappa) &= \mathrm{KL}(P_{pr}(x_{1:n_x})||q(x_{1:n_x};\kappa)) \\
&= \int P_{pr}(x_{1:n_x}) \log\left(\frac{P_{pr}(x_{1:n_x})}{q(x_{1:n_x};\kappa)}\right) dx \\
\kappa^* &= \operatorname*{argmin}_{\kappa} J(\kappa) \\
&= \operatorname*{argmin}_{\kappa} \mathbb{E}_{X\sim P_{pr}(x_{1:n_x})}[-\log(q(X = x_{1:n_x};\kappa))] \\
&= \operatorname*{argmax}_{\kappa} \mathbb{E}_{X\sim P_{pr}(x)}\left[\sum_{j=1, j\in S_r}^{n_x} \log(q^{in}_{a_j}(X_j = x_j|\phi_j;\eta_{a_j}))\right] \\
\eta^*_r &= \operatorname*{argmax}_{\eta_r} \mathbb{E}_{X\sim P_{pr}(x)}\left[\sum_{j=1}^{n_x} \mathbb{I}(a_j = r)\log(q^{in}_r(X_j = x_j|\phi_j;\eta_r))\right] \quad \forall r \in S_r, \\
\nabla_\kappa J(\kappa) &= \nabla_\kappa \mathbb{E}_{X\sim P_{pr}(x)}\left[\sum_{j=1}^{n_x} \log(q^{in}_{a_j}(X_j = x_j|\phi_j;\eta_{a_j}))\right] \\
&= \mathbb{E}_{X\sim P_{pr}(x)}\left[\sum_{j=1}^{n_x} \nabla_\kappa \log(q^{in}_{a_j}(X_j = x_j|\phi_j;\eta_{a_j})\right] \\
&\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{n_x} \nabla_\kappa \log(q^{in}_{a_j}(x^n_j|\phi_j;\eta_{a_j}))
\end{aligned}$$

Thus, we define the gradients to use for the stochastic gradient *ascent* for each subproblem $r \in S_r$ as:

$$\nabla_{\eta_r} J(\kappa) \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{n_x} \mathbb{I}(r = a_j)\nabla_{\eta_r} \log(q^{in}_r(x^n_j|\phi_j;\eta_r)) \tag{10}$$

where $x^n_{1:n_x} \overset{iid}{\sim} P_{pr}(x)$. During training we extract samples from each forward run and train each NSSP separately.

Appendix B. Details on the Regressor Training

To train our regressors, we use the L_2-norm $\mathbb{E}\left[\left\|R_r(\phi_j;\tau_r) - \hat{I}^{in}_r(\phi_j)\right\|^2_2\right]$ between our regressor $R_r(\phi_j;\tau_r)$ and our approximations of the marginal $\hat{I}^{in}_r(\phi_j)$. We then learn parameters τ_r so that it minimises this objective, resulting in $R_r(\phi_j;\tau_r) = I^{in}_r(\phi_j)$ in the limit of a large

number of training samples if our neural network has sufficient capacity to exactly capture $I_r^{in}(\phi_j)$. To see this note that

$$\mathbb{E}\left[\left\|R_r(\phi_j;\tau_r) - \hat{I}_r(\phi_j)\right\|_2^2\right] = \mathbb{E}\left[\left\|(R_r(\phi_j;\tau_r) - I_r(\phi_j)) + (I_r(\phi_j) - \hat{I}_r(\phi_j))\right\|_2^2\right] \tag{11}$$

$$= \mathbb{E}\left[\|R_r(\phi_j;\tau_r) - I_r(\phi_j)\|_2^2\right] + \mathbb{E}\left[\left\|I_r(\phi_j) - \hat{I}_r(\phi_j))\right\|_2^2\right] \tag{12}$$

where the second term does not depend on τ_r and the first is minimized when when $R_r(\phi_j;\tau_r) = I_{a_j}(\phi_j)$.

Our objective is defined as:

$$\mathcal{L}_r = \mathbb{E}_{\phi_j}\left\{\mathbb{E}_{\hat{I}_r^{in}}\left[\left\|R_r(\phi_j;\tau_r) - \hat{I}_r^{in}(\phi_j)\right\|_2^2 \;\middle|\; \phi_j\right]\right\} \tag{13}$$

$$\nabla_{\tau_r}\mathcal{L}_r = \mathbb{E}_{\phi_j}\left\{\mathbb{E}_{\hat{I}_r^{in}}\left[\nabla_{\tau_r}\left\|R_r(\phi_j;\tau_r) - \hat{I}_r^{in}(\phi_j)\right\|_2^2 \;\middle|\; \phi_j\right]\right\} \tag{14}$$

where the expectation over the inputs ϕ_j is defined by running P_{pr} forward and, if necessary, randomly selecting between the inputs that are passed to NSSP r if it is called more than once (this can further be Rao-Blackwellized by averaging over all the inputs passed to the NSSP instead of choosing between them). Thus, by running the simulator forward, collecting samples from the NSSPs generated from sampling the priors of each NSSP, we can make updates based on $\nabla_\tau(R_r(\phi_j;\tau) - \hat{I}_r(\phi_j))^2$ to minimise $\mathcal{L}_r$. With this approach, we must be careful to avoid over and under-fitting. Once trained, we can run inference on the approximate, unnormalised, target:

$$\hat{\gamma}(x_{1:n_x}) = \prod_{i=1,a_i\notin S_R}^{n_x} f_{a_i}(x_i;\phi_i) \prod_{k=1,b_k}^{n_y} g_{b_k}(y_k;\phi_k) \prod_{j=1,a_j\in S_R}^{n_x} \frac{\gamma_{a_j}^{in}(x_j|\phi_j)}{R_{a_j}(\phi_j;\tau)} \tag{15}$$

B.1. An Adjusted Approach for Nested Rejection Samplers

The approach outlined in Method 2 can be improved upon in the case where our nested sub-procedures are rejection samplers.

For rejection samplers, we always have $I(\phi) = \mathbb{E}[\mathbb{I}(A(z,\phi) = 1)]$ where $A(z,,\phi) = 1$ indicates an accepted sample and the expectation is with respect to running a single iteration of the rejection sampling loop. The naive Monte Carlo estimate for $I(\phi)$, $\frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(A(z_n,\phi) = 1)$, is only unbiased, if N is independent of the z_n.

Typically, one would like to instead run the rejection sampler in the standard manner, by which we generate samples by running the sampler until a sample is accepted, at which point we have generated N_a samples, where N_a is not independent of the z_n, such that the naive estimate is now biased. However, not doing this could, for example, return an estimate $\hat{(I)}(\phi) = 0$ which could cause significant issues if not dealt with properly, while it may not be possible to generate both strictly positive and unbiased estimates for $I(\phi)$.

This conundrum can be circumvented by instead trying to directly estimate $1/I(\phi)$ and use this as the basis for the regressor. This is possible because rejection samplers have the

property $\mathbb{E}[N_a|\phi] = 1/I(\phi)$ as follows:

$$\mathbb{E}[N_a|\phi] = \mathbb{E}\left[\sum_{n=1}^{N_a} 1 \middle| \phi\right] = \mathbb{E}\left[\sum_{n=1}^{\infty} \mathbb{I}(N_a \geq n) \middle| \phi\right] = \sum_{n=1}^{\infty} \mathbb{E}\left[\mathbb{I}(N_a \geq n)|\phi\right]$$
$$= \sum_{n=0}^{\infty} \mathbb{E}\left[\mathbb{I}(N_a > n)|\phi\right] = \sum_{n=0}^{\infty} (1 - I(\phi))^n = \frac{1}{I(\phi)}$$

Therefore, we learn our regressor $R'$to go from ϕ_j to $\mathbb{E}[N_a|\phi]$, exploiting the fact that N_a is an unbiased estimate of the latter, and subsequently use

$$P_{a_j}^{in}(x_j|\phi_j) \approx \gamma_{a_j}^{in}(x_j|\phi_j) R'_{a_j}(\phi_j; \tau_{a_j}) \tag{16}$$

to construct the approximate objective.

It is interesting to further note that

$$\mathbb{E}[\gamma_{a_j}^{in}(x_j|\phi_j) N_a | x_j, \phi_j] = P_{a_j}^{in}(x_j|\phi_j) \tag{17}$$

such that it should also be useful to use this result to develop pseudo-marginal samplers for such problems.

Appendix C. Experiment: Nested Gaussian NSSP

We take the model of a high-dimensional multivariate Gaussian with unkown mean and sample certain dimensions such that they rely on Gaussian NSSPs. The purpose of such an example is to demonstrate the validity of our methodoly, as it is one o the few examples in which we analytically calculate the correct ground truth. The model takes the following form:

$$\begin{aligned}
&\mu \leftarrow \mathbf{0} \\
&\mathbf{y} \leftarrow \{y_1, \ldots, y_d\} \\
&x_1 \sim \mathcal{N}(\mu_0, \Sigma_0) \\
&\texttt{for i = 2 : n} \\
&\quad \texttt{if i is odd} \\
&\quad\quad x_i \sim \texttt{innerprogram}(x_{i-1}) \\
&\quad \texttt{else :} \\
&\quad\quad x_i \sim \mathcal{N}(\mu_{i|i-1}, \Sigma_{i|i-1}) \\
&p(\mathbf{y}|\mathbf{x}) = \mathcal{MVN}(\mu_x, \hat{\Sigma}; \mathbf{y})
\end{aligned}$$

where the objective is to the mean of the multivariate normal (MVN). The (unnormalized) program density is given by:

$$p(x_{1:d}, y_{1:N}) = p(x_{1:d}) \prod_{i=1}^{N} p(y_i|x_{1:d})$$

Now if we choose Σ to be a valid covariance matrix with $\Sigma_{ij} = 0$ if $|i - j| \geq 2$, then we can write

$$p(x_{1:d}) = p(x_1)p(x_2|x_1) \ldots p(x_d|x_{d-1})$$

and we can sample $x_{1:d}$ sequentially from a Markov process. As the covariance matrix takes this structure we can use standard identities, as in Petersen et al., to analytically calculate the value of μ_x, which is plotted in Figure 1. Histograms both the predicted and ground truth values are provided in Figure 2 for all 60 dimensions.

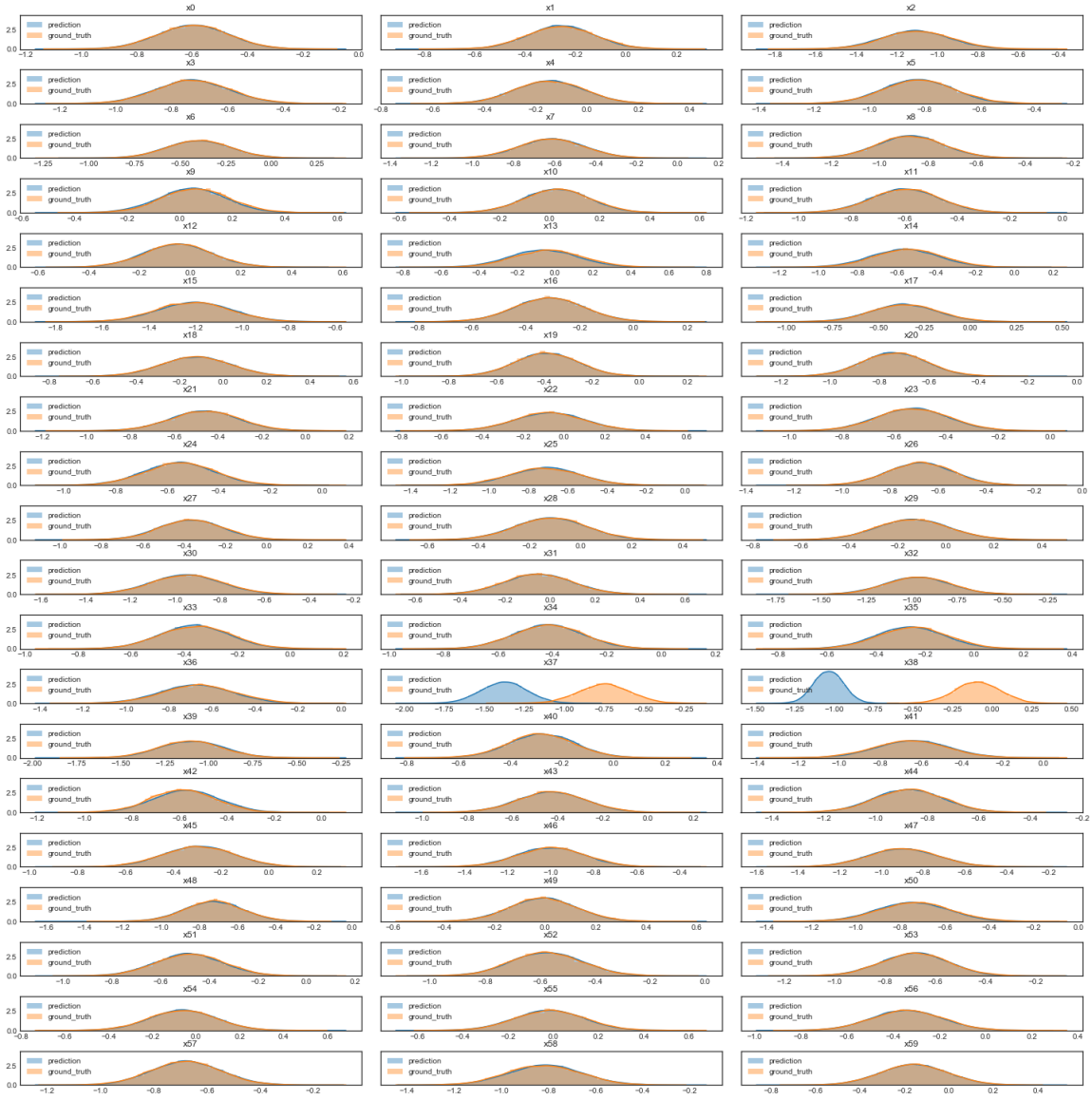

Figure 2: Histogram for each Latent variable, compared to the ground truth distribution for that latent variable.