Automatic Reparameterisation in Probabilistic Programming

Maria I. Gorinova[*] M.GORINOVA@ED.AC.UK
University of Edinburgh

Dave Moore DAVMRE@GOOGLE.COM
Matthew D. Hoffman MHOFFMAN@GOOGLE.COM
Google

Abstract

The performance of approximate posterior inference algorithms can depend strongly on how the model is parameterised. In particular, non-centring a model can break dependencies between different levels in hierarchical models and drastically reduce the difficulty of the inference task. However, it is not obvious how to reparameterise a model in the best possible way, as the shape of the posterior depends on the properties of the observed data.

We propose two inference strategies that utilise the power of probabilistic programming to free modellers from the need to choose a parameterisation. The first strategy alternates between sampling using the centred or the non-centred parameterisation, while the second strategy learns a partially non-centred parameterisation by optimising a variational objective.

1. Introduction

Reparameterising a probabilistic model means expressing it in terms of a different set of random variables, representing a bijective transformation of the original variables of interest. The reparameterised model can have drastically different posterior geometry from the original, with significant implications for both variational and sampling-based inference algorithms.

In this paper, we focus on *non-centring*, a particularly common form of reparameterisation in hierarchical Bayesian modelling. Consider a parameter $z \sim \mathcal{N}(\mu, \sigma)$; we say this is in *centred* parameterisation (CP). If we instead work with an auxiliary, standard normal parameter $\epsilon \sim \mathcal{N}(0, 1)$, and obtain z by applying the transformation $z = \mu + \sigma\epsilon$, we say the parameter is in its *non-centred* parameterisation (NCP).[1] Although the centred parameterisation is often more intuitive and interpretable, non-centring can sometimes drastically improve the performance of inference (Betancourt and Girolami, 2015). Figure 1 illustrates a simple example of such a case.

Bayesian practitioners are often advised to manually non-centre their models; however, this breaks the separation between modelling and inference and requires expressing the model in a potentially less intuitive form. Moreover, non-centring is not universally better than centring: the best parameterisation depends on many factors including the statistical properties of the observed data. The user must possess the sophistication to understand what reparameterisation is needed, and where in the model it should be applied.

We explore strategies to tackle this problem *automatically* via transformations of probabilistic programs. Using the Edward2 probabilistic programming language, we implement

[*] Work done while interning at Google.
1. More generally, non-centring is applicable to location-scale families and any random variable that can be expressed as a bijective transformation $z = f_\theta(\epsilon)$ of a "standardized" variable ϵ, analogous to the "reparameterisation trick" in stochastic optimisation (Kingma and Welling, 2013).

two approaches: Interleaved Hamiltonian Monte Carlo (i-hmc), which alternates HMC steps between centred and non-centred parameterisations, and a novel algorithm we call Variationally Inferred Parameterisation (vip), which uses a continuous relaxation to search over possible ways of reparameterising the model.[2] Experiments across a range of models demonstrate that these strategies enable robust inference, performing at least as well as the best fixed parameterisation, and sometimes better.

2. Automatic Model Reparameterisation

An advantage of probabilistic programming is that the program itself provides a structured model representation, and we can explore model reparameterisation through the lens of *program transformations*. Modern "deep" PPLs such as Pyro (Uber AI Labs, 2017) and Edward2 (Tran et al., 2018) provide built-in mechanisms for transforming a sampling process into an inference program, e.g., by overriding the behaviour of `sample` statements to instead compute the log-density of a given value under the sampling distribution. We will focus on Edward2, where this mechanism is called "interception" and can be understood as a special case of *effect handling* (Plotkin and Pretnar, 2009; Moore and Gorinova, 2018). In this work, we extend the usage of interception to treat `sample` statements in one parameterisation as `sample` statements in another parameterisation. For example, an NCP interceptor treats centred `sample` statements as non-centred. We give a more detailed explanation of interception in Appendix A, and present the interceptors used for this work in Appendix B.

We propose two strategies that tackle the reparameterisation automatically: Interleaved Hamiltonian Monte Carlo (i-hmc), and the Variationally Inferred Parameterisation (vip).

2.1. Interleaved Hamiltonian Monte Carlo

The Interleaved Hamiltonian Monte Carlo (i-hmc) algorithm uses two hmc steps to produce each sample from the target distribution. The first step is made in CP, using the original model parameters, while the second step is made in NCP, using the auxiliary standard parameters. The idea of interleaving MCMC kernels across parameterisations has been explored in previous work on interleaved Gibbs sampling (Yu and Meng, 2011; Kastner and Frühwirth-Schnatter, 2014), which demonstrated that we do not have to choose between CP and NCP, but can combine them to achieve more robust and performant samplers. Our contribution is that we make this interleaving automatic and model-agnostic: instead of requiring the user to write multiple versions of their model and a custom inference algorithm, we implement i-hmc in Tensorflow Probability, and use interceptors to make its usage only require the centred model definition. This brings i-hmc to the group of algorithms that can be used as a black-box algorithm as part of a probabilistic programming system.

Algorithm 1 outlines i-hmc in pseudo-code. The algorithm takes a single centred model $M_{cp}(\mathbf{z} \mid \mathbf{x})$ that defines parameters $\mathbf{z}$ and generates data $\mathbf{x}$. It then uses the function `make_ncp` to automatically obtain a non-centred version of the model, $M_{ncp}(\tilde{\mathbf{z}} \mid \mathbf{x})$, which defines the auxiliary standard variables $\tilde{\mathbf{z}}$, and the function f, such that $\mathbf{z} = f(\tilde{\mathbf{z}})$.

2. Code for these algorithms and experiments is available at https://github.com/google-research/google-research/tree/master/edward2_autoreparam.

3. Here we use HMC as the inference kernel. However, there is nothing specific to HMC in this strategy, so in practise we can use other any inference algorithm instead.

Algorithm 1: Interleaved Hamiltonian Monte Carlo

```
Arguments: data $\mathbf{x}$; a centred model $M_{cp}(\mathbf{z} \mid \mathbf{x})$
Returns: $S$ samples $\mathbf{z}^{(1)}, \ldots \mathbf{z}^{(S)}$ from $p(\mathbf{z} \mid \mathbf{x})$
1:  $M_{ncp}(\tilde{\mathbf{z}} \mid \mathbf{x}), f = \texttt{make\_ncp}(M_{cp}(\mathbf{z} \mid \mathbf{x}))$
2:  $\log p_{cp} = \texttt{make\_log\_joint}(M_{cp}(\mathbf{z} \mid \mathbf{x}))$
3:  $\log p_{ncp} = \texttt{make\_log\_joint}(M_{ncp}(\tilde{\mathbf{z}} \mid \mathbf{x}))$
4:
5:  $\mathbf{z}_0 = \texttt{init}()$
6:  for $s \in [1, \ldots, S]$ do
7:      $\mathbf{z}' = \texttt{hmc\_step}(\log p_{cp}, \mathbf{z}^{(s-1)})$
8:      $\mathbf{z}'' = \texttt{hmc\_step}(\log p_{ncp}, f^{-1}(\mathbf{z}'))$
9:      $\mathbf{z}^{(s)} = f(\mathbf{z}'')$;
10:
11: return $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(S)}$
```

Algorithm 2: Variationally Inferred Parameterisation

```
Arguments: data $\mathbf{x}$; a centred model $M_{cp}(\mathbf{z} \mid \mathbf{x})$
Returns: $S$ samples $\mathbf{z}^{(1)}, \ldots \mathbf{z}^{(S)}$ from $p(\mathbf{z} \mid \mathbf{x})$
1:  $M_{vip}(\tilde{\mathbf{z}} \mid \mathbf{x}; \boldsymbol{\phi}), f = \texttt{make\_vip}(M_{cp}(\mathbf{z} \mid \mathbf{x}))$
2:  $\log p(\mathbf{x}, \tilde{\mathbf{z}}) = \texttt{make\_log\_joint}(M_{vip}(\tilde{\mathbf{z}} \mid \mathbf{x}; \boldsymbol{\phi}))$
3:
4:  $Q(\tilde{\mathbf{z}}; \boldsymbol{\theta}) = \texttt{make\_variational}(M_{vip}(\tilde{\mathbf{z}} \mid \mathbf{x}; \boldsymbol{\phi}))$
5:  $\log q(\tilde{\mathbf{z}}; \boldsymbol{\theta}) = \texttt{make\_log\_joint}(q(\tilde{\mathbf{z}}; \boldsymbol{\theta}))$
6:
7:  $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_q(\log p(\mathbf{x}, \tilde{\mathbf{z}}; \boldsymbol{\phi})) - \mathbb{E}_q(\log q(\tilde{\mathbf{z}}; \boldsymbol{\theta}))$
8:  $\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \operatorname{argmax}_{\boldsymbol{\theta}, \boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$
9:  $\log p(\mathbf{x}, \tilde{\mathbf{z}}) = \texttt{make\_log\_joint}(M_{vip}(\tilde{\mathbf{z}} \mid \mathbf{x}; \boldsymbol{\phi}^*))$
10: $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(S)} = \texttt{hmc}(\log p)$[3]
11: return $f(\mathbf{z}^{(1)}), \ldots, f(\mathbf{z}^{(S)})$
```

2.2. Variationally inferred parameterisation

The best parameterisation for a given model may mix centred and non-centred representations for different variables. To efficiently search the space of reparameterisations, we construct a continuous relaxation of non-centring that includes both CP and NCP, and propose an algorithm VIP, which selects a parameterisation by gradient-based optimisation of a variational objective. VIP can be used as a pre-processing step to another inference algorithm. As it only changes the parameterisation of the model, using VIP in combination with an MCMC method does not have an effect on the asymptotic guarantees that the latter has.

Consider a model with parameters $\mathbf{z}$. We introduce a set of *parameterisation parameters* $\boldsymbol{\phi} = (\mathbf{a}, \mathbf{b})$, and transform each $z_i \sim \mathcal{N}(z_i \mid \mu_i, \sigma_i)$, by defining $\tilde{z}_i \sim \mathcal{N}(a_i \mu_i, \sigma_i^{b_i})$ and $z_i = \mu_i + \sigma_i^{1-b_i}(\tilde{z}_i - a_i \mu_i)$, where a_i and b_i are between 0 and 1. This parameterisation includes NCP as the special case $a = b = 0$, and CP as the case $a = b = 1$.

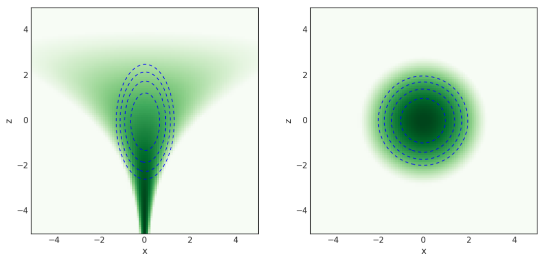
We learn a parameterisation by optimising an objective that favours parameterisations under which the posterior's shape is close to an independent normal distribution. A natural objective to choose is $\mathrm{KL}(q(\tilde{\mathbf{z}}; \boldsymbol{\theta}) \mid\mid p(\tilde{\mathbf{z}} \mid \mathbf{x}; \boldsymbol{\phi}))$, where $q(\tilde{\mathbf{z}}; \boldsymbol{\theta}) = \mathcal{N}(\tilde{\mathbf{z}} \mid \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$ is a mean-field variational model with *variational parameters* $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$. Minimising this divergence corresponds to maximimizing a variational lower bound (the ELBO):

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q(\tilde{\mathbf{z}}; \boldsymbol{\theta})} \left(\log p(\mathbf{x}, \tilde{\mathbf{z}}; \boldsymbol{\phi}) \right) - \mathbb{E}_{q(\tilde{\mathbf{z}}; \boldsymbol{\theta})} \left(\log q(\tilde{\mathbf{z}}; \boldsymbol{\theta}) \right).$$

Neal's funnel (Figure 1) provides an illustrative example: the variational distribution is a poor fit to the pathological geometry of CP, but non-centring leads to a perfect fit. We simultaneously fit the variational distribution $q(\tilde{\mathbf{z}}; \boldsymbol{\theta})$ to the posterior $p(\tilde{\mathbf{z}} \mid \mathbf{x}; \boldsymbol{\phi})$ by changing the variational parameters $\boldsymbol{\theta}$, and change the shape of that posterior through the parameterisation parameters $\boldsymbol{\phi}$. Algorithm 2 summarises the method. Both the model reparameterisation and the construction of the variational distribution q are done automatically, utilising Edward2's interceptors (see Appendix B).

3. Experiments

We report experimental results for hierarchical Bayesian regression on the Eight Schools (Rubin, 1981), Radon (Gelman and Hill, 2006) and German credit datasets. For each, we specify a model and evaluate I-HMC and VIP-HMC by comparing to HMC run on the fully



(a) Centred (b) Non-centred

Figure 1: Neal's funnel: $z \sim N(0,3)$; $x \sim N(0, e^{-z/2})$ (Neal, 2003), with mean-field normal variational fit overlayed. Although half of the mass is inside the "funnel" ($z < 0$), centred samplers have difficulty reaching it.

	8 Schools	**German Credit**
HMC-CP	92 $\pm$4	64 $\pm$ 21
HMC-NCP	3475 $\pm$ 849	35 $\pm$ 15
I-HMC	3879 $\pm$ 281	101 $\pm$ 32
VIP-HMC	4986 $\pm$ 660	115 $\pm$ 12

Table 1: Effective sample size to number of leapfrog steps (larger is better), with standard errors from five trials. Eight schools and German credit models.

centred model (HMC-CP) and the fully non-centred model (HMC-NCP). On each run of the experiment, we obtain 50000 samples after burn-in. We tune the step sizes and number of leapfrog steps for HMC automatically, as described in more details in Appendix C.

	MN	**IN**	**PA**	**MO**	**ND**	**MA**	**AZ**
HMC-CP	798 ± 276	1034 ± 54	425 ± 208	427 ± 45	2840 ± 347	5796 ± 393	3644 ± 129
HMC-NCP	340 ± 35	75 ± 14	43 ± 9	16 ± 7	187 ± 36	179 ± 66	100 ± 26
I-HMC	1495 ± 129	590 ± 287	410 ± 183	233 ± 29	2421 ± 89	6696 ± 97	2472 ± 267
VIP-HMC	1144 ± 279	865 ± 98	816 ± 184	416 ± 51	3273 ± 145	5551 ± 336	3875 ± 73

Table 2: Effective sample size to number of leapfrog steps (larger is better), with standard errors from five trials. Radon data for different US states.

Across the datasets in Table 1 and Table 2, we see that I-HMC is a robust alternative to using a fully centred or non-centred model. By taking alternating steps in CP and NCP, I-HMC ensures that it makes reasonable progress, regardless of which of HMC-CP or HMC-NCP is better. Moreover, we see that VIP-HMC finds a reasonable reparameterisation in each case; typically as good as of the better of HMC-CP and HMC-NCP. On initial inspection, the learned parameterisations are often very close to fully centred or non-centred (implying that VIP-HMC successfully learns the "correct" global parameterisation for each problem), but a small number of groups are sometimes flipped to the alternative parameterisation. These preliminary results suggest that these learned, mixed parameterisations may sometimes be superior to the best global parameterisation; we are excited to explore this further.

4. Discussion

We presented two inference strategies that use program transformations on probabilistic programs to automatically make use of different model reparameterisations, and we show both strategies are robust. We hope that the idea of automatically reparameterising probabilistic models with the aid of program transformations can lead to new ways of easing the inference task, potentially allowing us to work with models that were previously infeasable.

References

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and computing*, 18(4):343–373, 2008.

Michael Betancourt and Mark Girolami. Hamiltonian Monte Carlo for hierarchical models. *Current trends in Bayesian methodology with applications*, 79:30, 2015.

Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2006.

Gregor Kastner and Sylvia Frühwirth-Schnatter. Ancillarity-sufficiency interweaving strategy (ASIS) for boosting MCMC estimation of stochastic volatility models. *Computational Statistics & Data Analysis*, 76:408–423, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Dave Moore and Maria I. Gorinova. Effect handling for composable program transformations in Edward2. *International Conference on Probabilistic Programming*, 2018. URL https://arxiv.org/abs/1811.06150.

Radford M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–741, 2003. ISSN 00905364. URL http://www.jstor.org/stable/3448413.

Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In Giuseppe Castagna, editor, *Programming Languages and Systems*, pages 80–94, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-00590-9.

Matija Pretnar. An introduction to algebraic effects and handlers. Invited tutorial paper. *Electronic Notes in Theoretical Computer Science*, 319:19 – 35, 2015. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2015.12.003. URL http://www.sciencedirect.com/science/article/pii/S1571066115000705. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

Donald B. Rubin. Estimation in parallel randomized experiments. *Journal of Educational Statistics*, 6(4):377–401, 1981. ISSN 03629791. URL http://www.jstor.org/stable/1164617.

Dustin Tran, Matthew D. Hoffman, Srinivas Vasudevan, Christopher Suter, Dave Moore, Alexey Radul, Matthew Johnson, and Rif A. Saurous. Simple, Distributed, and Accelerated probabilistic programming. 2018. URL https://arxiv.org/abs/1811.02091. To appear in Advances in Neural Information Processing Systems 2019.

Uber AI Labs. Pyro: A deep probabilistic programming language, 2017. http://pyro.ai/.

Yaming Yu and Xiao-Li Meng. To center or not to center: That is not the question—an ancillarity–sufficiency interweaving strategy (ASIS) for boosting MCMC efficiency. *Journal of Computational and Graphical Statistics*, 20(3):531–570, 2011.

Appendix A. Probabilistic Programming in Edward2

Edward2 (Tran et al., 2018) is a deep probabilistic programming language built on top of TensorFlow, which similarly to Pyro (Uber AI Labs, 2017) uses *algebraic effect handlers*(Plotkin and Pretnar, 2009; Pretnar, 2015) to transform a generative model to a target function on the parameters of the model, that can be used for inference. This section outlines the basic concepts of probabilistic programming in Edward2.

A model in Edward2 is a Python function that generates random variables. For example, the following function corresponds to Neal's funnel from Figure 1:

```
from tensorflow_probability import edward2 as ed
def neals_funnel():
    z = ed.Normal(loc=0., scale=3., name="z")
    x = ed.Normal(loc=0., scale=tf.exp(-z / 2.), name="x")
    return x
```

When run forward, this generates samples for `x`. However, in most cases when the task is to sample from some posterior distribution given data, we are interested in a function that evaluates the density, rather than the generative model. Edward2 uses a mechanism called *interception* to do that.

Interception can be seen as a special case of effect handling, and the idea behind it is that it treats the construction of random variables as an *effectful* operation. Effectful operations are operations that can potentially have some side effect (e.g. writing to a systems file). Effectful operations can be *intercepted* (or in PL jargon they are *handled*), so that their effect can be controlled.

Continuing with Neal's funnel, we can define a function that evaluates the log density $\log p(x, z)$ at some given x and z:

```
def log_joint_fn(**kwargs):
  log_prob = 0

  def log_prob_interceptor(rv_constructor, **rv_kwargs):
    # Overrides a random variable's ‘value‘ and accumulates its log-prob.
    rv_name = rv_kwargs.get("name")
    rv_kwargs["value"] = kwargs.get(rv_name)

    rv = rv_constructor(**rv_kwargs)
    log_prob = log_prob + rv.distribution.log_prob(rv.value)
    return rv

  with ed.interception(log_prob_interceptor):
    neals_funnel()

  return log_prob
```

By executing the `neals_funnel` function in the context of `log_prob_interceptor`, we override each sample statement (a call to a random variable constructor `rv_constructor`), to generate a variable that takes on the value provided in the arguments of `log_joint_fn`. As a side effect, we also accumulate the result of evaluating each variable's prior density at the provided value, which, by the chain rule, gives us the log joint density. The function `log_joint_fn` then is equivalent to the function $\log p$, where $\log p(z, x) = \log \mathcal{N}(z \mid 0, 1) + \log \mathcal{N}(x \mid 0, e^{-z/2})$.

Appendix B. Interceptors

Interceptors can be used as a powerful abstractions in a probabilistic programming systems, as discussed previously by Moore and Gorinova (2018), and shown by both Pyro and Edward2. In particular, we can use interceptors to automatically reparameterise a model, as well as to specify variational families. In this section, we show Edward2 pseudo-code for the interceptors used to implement i-hmc and vip-hmc.

B.1. Non-centred Parameterisation Interceptor

By intercepting every construction of a normal variable, [4] we can create a standard normal variable instead, and scale and shift appropriately.

```
def ncp_interceptor(rv_constructor, **rv_kwargs):
    # Assumes rv_constructor is in the location-scale family
    name = rv_kwargs["name"] + "_std"
    rv_std = ed.interceptable[5](rv_constructor)(loc=0, scale=1)
    return rv_kwargs["loc"] + rv_kwargs["scale"] * rv_std
```

Running a model that declares the random variables $\boldsymbol{\theta}$ in the context of `ncp_interceptor` will declare a new set of standard normal random variables $\boldsymbol{\theta}^{(\mathrm{std})}$. Nesting this in the context of the `log_prob_interceptor` from Appendix A will then evaluate the log joint density $\log p(\boldsymbol{\theta}^{(\mathrm{std})})$.

For example, going back to Neal's funnel, running

```
with ed.interception(log_prob_interceptor):
    neals_funnel()
```

corresponds to evaluating $\log p(z, x) = \log \mathcal{N}(z \mid 0, 1) + \log \mathcal{N}(x \mid 0, e^{-z/2})$, while running

```
with ed.interception(log_prob_interceptor):
    with ed.interception(ncp_interceptor):
        neals_funnel()
```

corresponds to evaluating $\log p(z^{(\mathrm{std})}, x^{(\mathrm{std})}) = \log \mathcal{N}(z^{(\mathrm{std})} \mid 0, 1) + \log \mathcal{N}(x^{(\mathrm{std})} \mid 0, 1)$.

B.2. VIP Interceptor

The VIP interceptor is similar to the NCP interceptor. The notable difference is that it creates new learnable Tensorflow variables, which correspond to the parameterisation parameters a and b:

```
def vip_interceptor(rv_constructor, **rv_kwargs):
    name = rv_kwargs["name"] + "_vip"
    rv_loc = rv_kwargs["loc"]
    rv_scale = rv_kwargs["scale"]

    a = tf.nn.sigmoid(tf.get_variable(
      name + "_a_unconstrained", initializer=tf.zeros_like(rv_loc))
    b = tf.nn.sigmoid(tf.get_variable(
      name + "_b_unconstrained", initializer=tf.zeros_like(rv_scale))
```

4. Or, more generally, of location-scale family variables.
5. Wrapping the constructor in with `ed.interceptable` ensures that we can nest this interceptor in the context of other interceptors.

```
    rv_vip = ed.interceptable(rv_constructor)(
                loc=a * rv_loc, scale=b ** rv_scale)
    return rv_loc + rv_scale ** (1 - b) * (rv_vip - a * rv_loc)
```

B.3. Mean-field Variational Model Interceptor

Finally, we show a mean-field variational familiy interceptor, which we use both to tune the step sizes for HMC (see Appendix C), and to make use of VIP automatically. The `mfvi_interceptor` simply substitutes each sample statement with sampling from a normal distribution with parameters specified by some fresh variational parameters μ and σ:

```
def vip_interceptor(rv_constructor, **rv_kwargs):
    name = rv_kwargs["name"] + "_q"
    mu = tf.get_variable(name + "_mu")
    sigma = tf.nn.softmax(tf.get_variable(name + "_sigma"))

    rv_q = ed.interceptable(ed.Normal)(
            loc=mu, scale=sigma, name=name)
    return rv_q
```

Appendix C. Experimental Setup

Algorithms. We evaluate I-HMC and VIP-HMC by comparing to HMC run on a fully centred model (HMC-CP) and fully non-centred model (HMC-NCP). On each run of the experiment, we obtain 50000 samples, in addition to the first 8000 samples that we discard as burn-in. We also thin each HMC chain by discarding every other sample, in order to match the work done per sample to that of I-HMC (where the intermediate NCP sample is discarded).

Tuning. We run each algorithm using HMC with 1, 2, 4, 8, 16, and 32 leapfrog steps, and report the results for each model and each dataset that maximize (effective sample size / leapfrog steps), where the joint effective sample size is taken to be the minimum across all model variables. Within each HMC chain, the per-variable step sizes are initialized using the posterior standard deviation computed by mean-field variational inference (MF-VI). The step-size is then adapted for the first 6000 steps of the burnin using a procedure following Section 4.2 of Andrieu and Thoms (2008), targeting an acceptance rate of 0.75. In the case of I-HMC, we use fixed step sizes for each sub-step: the resulting step sizes after adaptation of HMC-CP and HMC-NCP for the CP and NCP sub-steps respectively. As this approximation is noisy, yet different values can drastically change the quality of the inferred samples, we approximate and adapt the step sizes at every experimental run, for every algorithm.

Variational Inference Setup. The MF-VI procedure uses 64 samples from the variational distribution q for a Monte Carlo estimate of the expectations under q. We optimise the ELBO using Adam optimiser. To reduce variance, we run the MF-VI procedure 3 times using learning rate of 0.01 and 3 times using learning rate of 0.1, and chose the result that maximizes the ELBO.