Automatic Reparameterisation in Probabilistic Programming

Maria I. Gorinova

University of Edinburgh

Dave Moore
Google

Matthew D. Hoffman
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).

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 $\epsilon$, 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.\footnote{Code for these algorithms and experiments is available at \url{https://github.com/google-research/google-research/tree/master/edward2_autoreparam}.} 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 \textit{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 \texttt{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 \textit{effect handling} (Plotkin and Pretnar, 2009; Moore and Gorinova, 2018). In this work, we extend the usage of interception to treat \texttt{sample} statements in one parameterisation as \texttt{sample} statements in another parameterisation. For example, an NCP interceptor treats centred \texttt{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 \texttt{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 $\mathbf{z} = f(\tilde{\mathbf{z}})$.

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.
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 $z$. We introduce a set of parameterisation parameters $\phi = (a, b)$, 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 $\text{KL}(q(\tilde{z}; \theta) \mid \mid p(\tilde{z} \mid x; \phi))$, where $q(\tilde{z}; \theta) = \mathcal{N}(\tilde{z} \mid \mu, \text{diag}(\sigma))$ is a mean-field variational model with variational parameters $\theta = (\mu, \sigma)$. Minimising this divergence corresponds to maximising a variational lower bound (the ELBO):

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

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 $q(\tilde{z}; \theta)$ to the posterior $p(\tilde{z} \mid x; \phi)$ by changing the variational parameters $\theta$, and change the shape of that posterior through the parameterisation parameters $\phi$. 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
Automatic Reparameterisation

![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.](image)

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.

<table>
<thead>
<tr>
<th></th>
<th>8 Schools</th>
<th>German Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC-CP</td>
<td>92 ± 4</td>
<td>64 ± 21</td>
</tr>
<tr>
<td>HMC-NCP</td>
<td>3475 ± 849</td>
<td>35 ± 15</td>
</tr>
<tr>
<td>1-HMC</td>
<td>3879 ± 281</td>
<td>101 ± 32</td>
</tr>
<tr>
<td>VIP-HMC</td>
<td>4986 ± 660</td>
<td>115 ± 12</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th></th>
<th>MN</th>
<th>IN</th>
<th>PA</th>
<th>MO</th>
<th>ND</th>
<th>MA</th>
<th>AZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC-CP</td>
<td>798 ± 276</td>
<td>1034 ± 54</td>
<td>425 ± 208</td>
<td>427 ± 45</td>
<td>2840 ± 347</td>
<td>5796 ± 393</td>
<td>3644 ± 129</td>
</tr>
<tr>
<td>HMC-NCP</td>
<td>340 ± 35</td>
<td>75 ± 14</td>
<td>43 ± 9</td>
<td>16 ± 7</td>
<td>187 ± 36</td>
<td>179 ± 66</td>
<td>100 ± 26</td>
</tr>
<tr>
<td>1-HMC</td>
<td>1495 ± 129</td>
<td>590 ± 287</td>
<td>410 ± 183</td>
<td>233 ± 29</td>
<td>2421 ± 89</td>
<td>6696 ± 97</td>
<td>2472 ± 267</td>
</tr>
<tr>
<td>VIP-HMC</td>
<td>1144 ± 279</td>
<td>865 ± 98</td>
<td>816 ± 184</td>
<td>416 ± 51</td>
<td>3273 ± 145</td>
<td>5551 ± 336</td>
<td>3875 ± 73</td>
</tr>
</tbody>
</table>

Across the datasets in Table 1 and Table 2, we see that 1-HMC is a robust alternative to using a fully centred or non-centred model. By taking alternating steps in CP and NCP, 1-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


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:

```python
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 $z$:

```python
def log_joint_fn(**kwargs):
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, we can create a standard normal variable instead, and scale and shift appropriately.

```python
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(rv_constructor)(loc=0, scale=1)
    return {k: v*r*rv_std for k, v in rv_kwargs.items()}
```

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

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

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

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

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

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

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 $b$:

```python
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 family 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.