An Adaptive Resample-Move Algorithm for Estimating Normalizing Constants

Marco Fraccaro[a], Ulrich Paquet, Ole Winther[a,*]

[a]*Technical University of Denmark, Lyngby, Denmark*

Abstract

The estimation of normalizing constants is a fundamental step in probabilistic model comparison. Sequential Monte Carlo methods may be used for this task and have the advantage of being inherently parallelizable. However, the standard choice of using a fixed number of particles at each iteration is suboptimal because some steps will contribute disproportionately to the variance of the estimate. We introduce an adaptive version of the Resample-Move algorithm, in which the particle set is adaptively expanded whenever a better approximation of an intermediate distribution is needed. The algorithm builds on the expression for the optimal number of particles and the corresponding minimum variance found under ideal conditions. Benchmark results on challenging Gaussian Process Classification and Restricted Boltzmann Machine applications show that Adaptive Resample-Move (ARM) estimates the normalizing constant with a smaller variance, using less computational resources, than either Resample-Move with a fixed number of particles or Annealed Importance Sampling. A further advantage over Annealed Importance Sampling is that ARM is easier to tune.

Keywords: Sequential Monte Carlo, resample-move, Riemannian manifold Hamiltonian Monte Carlo, estimating normalizing constants, estimating partition functions

1. Introduction

The optimization of computational resources in Sequential Monte Carlo (SMC) algorithms is an open research problem. Recent efforts to parallelize and distribute implementations represent a possible solution [1, 2]. In this work we approach this issue from a different perspective, showing that SMC methods can also be made more efficient by using less particles, if they are used optimally. While a few particles are needed to provide an accurate empirical estimate of an intermediate distribution, in other "high variance" iterations more reliable results can be obtained if the particle set is extended to better approximate expectations of interest. Our proposed algorithm, *Adaptive Resample-Move*, represents a theoretically grounded way to exploit this idea for optimizing the estimation of normalizing constants.

Under a fixed computational budget, the optimal way to minimize the variance of the estimate of the normalizing constant is to use a variable number of particles at each iteration. This is proved in Section 2 for the ideal condition of independent samples. This result is then used to define *Adaptive Resample-Move* (ARM), an extension of an SMC method known as Resample-Move [3]. ARM finds accurate estimates using an adaptive number of particles at each iteration (see Section 3). Experimentally we show that, from a computational view, it is better to adaptively grow the number of particles per iteration as needed.

The proposed algorithm is compared to state of the art methods on two sets of challenging machine learning problems: In Section 4, ARM is compared to several versions of Annealed Importance Sampling [4] for Gaussian Process (GP) Classification. To obtain a competitive baseline, we derived a Riemannian Manifold Hamiltonian Monte Carlo [5] sampler for GP models, which would be of independent interest [6]. Due to the importance of Restricted Boltzmann Machines to the deep learning community, we evaluated ARM in Section 5 to estimate their normalizing constants. The results indicate that ARM provides very competitive accuracy at a lower computational cost and perhaps most importantly with more ease for adapting the interpolation to the problem at hand. There exists a large body of related work, which we describe in the context of this paper in Section 6.

2. The Resample-Move algorithm

Sequential Monte Carlo (SMC) algorithms [7] obtain samples from a target distribution p by iteratively sampling from a sequence of distributions $p_1, p_2, \ldots, p_N = p$. Although SMC is more generally applicable, we restrict ourselves to a sequence of distributions

$$p_n(\mathbf{x}_{[n]}) = \frac{1}{Z_n} f_n(\mathbf{x}_{[n]})$$

that are defined on spaces of increasing dimensionality, where $\mathbf{x}_{[n]} = (x_1, x_2, \ldots, x_n)$ indicates the vector of the first n components of $\mathbf{x}$, and

$$Z_n = \int f_n(\mathbf{x}_{[n]})\, \mathrm{d}\mathbf{x}_{[n]} \,.$$

Note that each consecutive f_n has *no* dependence on variables x_i for $i > n$.

*Corresponding author

Email addresses: marfra@dtu.dk (Marco Fraccaro), ulrich@cantab.net (Ulrich Paquet), olwi@dtu.dk (Ole Winther)

Algorithm 1 Resample-Move

1: $x_1^{[R]} \sim p_1(x_1)$; $\widetilde{w}_1^{[R]} := \frac{1}{R}$; $\log Z := \log Z_1$
2: **for** $n = 1$ to $N-1$ **do**
3: $\quad \mathbf{x}_{[n]}^{[R]} \sim \mathcal{K}(\mathbf{x}_{[n]}; \mathbf{x}_{[n]}^{[R]})$ *// move*
4: $\quad w_{n+1}^{[R]} := \mathcal{W}(\mathbf{x}_{[n]}^{[R]})\, \widetilde{w}_n^{[R]}$ *// smooth*
5: $\quad \log Z := \log Z + \log \sum_r w_{n+1}^r$
6: $\quad \widetilde{w}_{n+1}^{[R]} := w_{n+1}^{[R]} / \sum_i w_{n+1}^i$
7: $\quad$ **if** $R_{\text{eff}}(\widetilde{w}_{n+1}^{[R]}) < R_{\text{eff}}^{\min}$ **then**
8: $\quad\quad \mathbf{x}_{[n]}^{[R]} := \text{resample}(\widetilde{w}_{n+1}^{[R]}, \mathbf{x}_{[n]}^{[R]})$ *// resample*
9: $\quad\quad \widetilde{w}_{n+1}^{[R]} := \frac{1}{R}$
10: $\quad$ **end if**
11: $\quad x_{n+1}^{[R]} \sim x_{n+1} | \mathbf{x}_{[n]}^{[R]}$ *// augment*
12: **end for**
13: **return** $\log Z$ and $\mathbf{x}_{[N]}^{[R]}$

At iteration n, where the iterations run $n = 1, \ldots, N$, a set of R particles $\mathbf{x}_{[n]}^{[R]} = (\mathbf{x}_{[n]}^1, \mathbf{x}_{[n]}^2, \ldots, \mathbf{x}_{[n]}^R)$ with weights $w_n^{[R]} = (w_n^1, w_n^2, \ldots, w_n^R)$ are kept, such that they provide an empirical estimate of p_n, in the sense that

$$\sum_{r=1}^R \widetilde{w}_n^r \, \varphi(\mathbf{x}_{[n]}^r) \to \mathbb{E}_n[\varphi(\mathbf{x}_{[n]})] \tag{1}$$

almost surely as $R \to \infty$, for any measurable φ such that the expectation $\mathbb{E}_n[\varphi(\mathbf{x}_{[n]})]$ exists. Notation $\widetilde{w}_n^r = w_n^r / \sum_{r'=1}^R w_n^{r'}$ indicates the normalized weight of the r'th particle, and $\mathbb{E}_n$ is a shorthand for the expectation $\mathbb{E}_{p_n(\mathbf{x}_{[n]})}$. If (1) holds, we say that $(\mathbf{x}_{[n]}^{[R]}, w_n^{[R]})$ *targets* $p_n(\mathbf{x}_{[n]})$. The target of the particle system evolves over time: samples from $p_{n+1}(\mathbf{x}_{[n+1]})$ are obtained with importance sampling and resampling techniques using $p_n(\mathbf{x}_{[n]})$ as a proposal distribution.

The normalizing constant $Z = Z_N$ of the target distribution unrolls over the sequence with

$$\begin{aligned}
\log Z &= \log Z_1 + \sum_{n=1}^{N-1} \log \frac{Z_{n+1}}{Z_n} \\
&= \log Z_1 + \sum_{n=1}^{N-1} \log \frac{1}{Z_n} \int \frac{f_{n+1}(\mathbf{x}_{[n+1]})}{f_n(\mathbf{x}_{[n]})} f_n(\mathbf{x}_{[n]}) \, \mathrm{d}\mathbf{x}_{[n+1]} \\
&= \log Z_1 + \sum_{n=1}^{N-1} \log \mathbb{E}_n \left[\int \frac{f_{n+1}(\mathbf{x}_{[n+1]})}{f_n(\mathbf{x}_{[n]})} \, \mathrm{d}x_{n+1} \right].
\end{aligned} \tag{2}$$

A recursive unbiased estimate of $\log Z$ can then be obtained using the set of particles that target $p_n(\mathbf{x}_{[n]})$ to approximate the expectation in (2) with a weighted average. Starting at x_1, the number of variables averaged over is therefore sequentially increased by one with the Resample-Move algorithm outlined in Algorithm 1, which uses any random ordering of variables to decompose $\log Z$. The algorithm's key steps are illustrated in Figure 1 and discussed in detail in Section 2.1.

2.1. Resample-Move

Resample-Move (RM) [3] extends standard SMC methods using an MCMC kernel to increase diversity in the particles (see also Section 2.2). It can be seen as a special case of the very general SMC framework introduced in [8]. As invariant precondition to Line 3's move-step, $(\mathbf{x}_{[n]}^{[R]}, \widetilde{w}_n^{[R]})$ targets p_n, and this property is retained after application of a Markov chain Monte Carlo (MCMC) transition kernel $\mathcal{K}$ that has $p_n(\mathbf{x}_{[n]})$ as invariant distribution. The smooth-step in Line 4 computes the smoothed ratio

$$\mathcal{W}(\mathbf{x}_{[n]}) = \frac{p_{n+1}(\mathbf{x}_{[n]})}{p_n(\mathbf{x}_{[n]})} = \int \frac{f_{n+1}(\mathbf{x}_{[n+1]})}{f_n(\mathbf{x}_{[n]})} \, \mathrm{d}x_{n+1}, \tag{3}$$

by marginalizing out x_{n+1}. Notation $p_{n+1}(\mathbf{x}_{[n]}) \doteq \int p_{n+1}(\mathbf{x}_{[n+1]}) \, \mathrm{d}x_{n+1}$ denotes the *marginal* distribution of $p_{n+1}(\mathbf{x}_{[n+1]})$. With the updated importance weights being $w_{n+1}^r = \mathcal{W}(\mathbf{x}_{[n]}^r)\widetilde{w}_n^r$, the set $(\mathbf{x}_{[n]}^{[R]}, \widetilde{w}_{n+1}^{[R]})$ will target $p_{n+1}(\mathbf{x}_{[n]})$. Note that $\mathcal{W}(\mathbf{x}_{[n]})$ is exactly the argument of the expectation in (2); it is hence possible to update the incremental estimate of $\log Z$ (Line 5) using the Monte Carlo approximation in (1). To ensure that Line 3's precondition will hold in the next iteration, the augmentation-step in Line 11 adds a component x_{n+1}^r to each particle by sampling from the conditional distribution $x_{n+1}^r \sim p_{n+1}(x_{n+1} | \mathbf{x}_{[n]}^r)$. The particles $(\mathbf{x}_{[n+1]}^{[R]}, \widetilde{w}_{n+1}^{[R]})$ will then target $p_{n+1}(\mathbf{x}_{[n+1]})$.

The resample-step in Lines 8 and 9 is wedged between the smooth- and augmentation-steps, and returns a new set of uni-

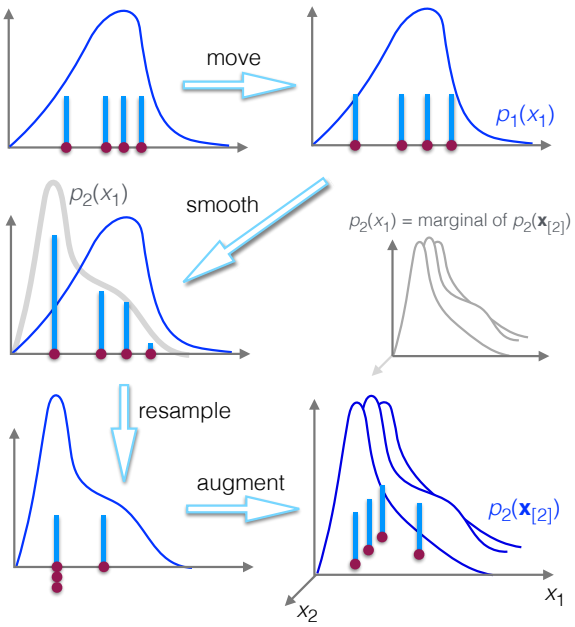

Figure 1: A sketch of the first iteration of Resample-Move in Algorithm 1, starting with four weighted particles $x_1^{[4]}$ that target $p_1(x_1)$. The heights of the bars reflect the particle weights $w_1^{[4]}$ (top left). After application of a MCMC transition kernel $\mathcal{K}(x_1; x_1^{[4]})$ in the **move**-step, the moved particles $x_1^{[4]}$ still target $p_1(x_1)$. In the **smooth**-step, the weights of the particles are adjusted to $w_2^{[4]}$, so that they target the marginal distribution $p_2(x_1) \doteq \int p_2(\mathbf{x}_{[2]}) \, \mathrm{d}x_2$. The implication of this step is that one can extend each particle by a new dimension in the **augment**-step, by sampling x_2^r from $p_2(x_2 | x_1^r)$ for $r = 1, \ldots, 4$. Their weights are kept unchanged. The newly augmented (weighted) particles target $p_2(\mathbf{x}_{[2]})$. Before the augment-step, one might want to reset the weights $w_2^{[4]}$ on the set $x_1^{[4]}$ to be uniform. This is done by stochastically replicating high-weighted particles under $p_2(x_1)$ in the **resample**-step.

formly weighted particles that are resampled from the old ones using weights $\widetilde{w}_{n+1}$. Multinomial or residual resampling [9] are commonly used.[1] After Line 9, the set still targets $p_{n+1}(\mathbf{x}_{[n]})$, and is formed in such a way that the most significant particles are repeated to serve as multiple starting points for the next move-step, while particles with low weights are discarded.

Resampling should be done only if necessary to prevent the *degeneracy* problem (that is, when $\widetilde{w}_{n+1}^{[R]}$ is such that only few particles have a significant weight, and all the others have a very small weight), as resampling introduces correlation among particles and additional variance in the estimates. It is common to use the *Effective Sample Size* (ESS) [9]

$$R_{\text{eff}}(w_{n+1}^{[R]}) = \frac{(\sum_{r=1}^{R} w_{n+1}^{r})^2}{\sum_{r=1}^{R}(w_{n+1}^{r})^2} = \frac{1}{\sum_{r=1}^{R}(\widetilde{w}_{n+1}^{r})^2} \tag{4}$$

as a yardstick to measure the number of particles with a significantly high weight. The resample-step is then done only if the ESS is below a certain threshold, for example $R_{\text{eff}}^{\min} = 0.7R$.

2.2. Shortcomings of RM and other SMC algorithms

The resample-step is a powerful way to deal with the degeneracy problem, as after the resample-step, all particles have an equal weight. However, it introduces a new issue, namely *sample impoverishment*. Particles with a high weight are likely to be resampled many times, and this means that the actual number of particles contributing to the weighted average in (1) may be much smaller than R. RM reduces sample impoverishment with the move-step, which increases diversity in the particles. These additional steps are useful only if any two consecutive sequential distributions are similar enough: in our case $p_n(\mathbf{x}_{[n]})$ has to be reasonably close to $p_{n+1}(\mathbf{x}_{[n]})$; see (3) and Figure 1. If they differ too much, only a few particles might suddenly be significant in the next iteration (in the worst case no particles would fall in the high probability density region), making it very difficult for the system to provide again a good approximation of any expectation of interest. In sequential parameter estimation, for example, this could happen when a particularly difficult data observation is to be introduced [11], and the distribution changes to one that is very different. Section 2.3 proves a theorem that under a fixed computational budget R_{tot}, the difference between distributions translates to RM requiring a higher number of particles R_n in iteration n.

2.3. Optimal number of particles at each iteration

We argued that if $p_n(\mathbf{x}_{[n]})$ and $p_{n+1}(\mathbf{x}_{[n]})$ are not similar enough, then a higher number of particles is needed, and formalize the statement here. Let the number of particles at each iteration be variable, so that $p_n(\mathbf{x}_{[n]})$ is approximated with R_n particles. Given a computational budget of $R_{\text{tot}} = \sum_{n=1}^{N-1} R_n$, we may wonder what the optimal values for R_n for $n = 1, \dots N-1$ are, such that the variance of the estimate of the normalizing constant

$$\log \widehat{Z} = \log Z_1 + \textstyle\sum_{n=1}^{N-1} \log\big(\sum_{r=1}^{R_n} \mathcal{W}(\mathbf{x}_{[n]}^r)\widetilde{w}_n^r\big) \tag{5}$$

is minimized (see (2)). The following theorem gives the answer to this question under ideal conditions. Let $\mathbb{V}_n[\cdot]$ denote the variance of its argument under $p_n(\mathbf{x}_{[n]})$.

Theorem 1. *Assume independent equally-weighted samples from the distributions $p_n(\mathbf{x}_{[n]})$ for $n = 1, \dots N-1$, and define the variance of the normalized weight updates*

$$v_n \doteq \mathbb{V}_n\left[\frac{\mathcal{W}(\mathbf{x}_{[n]})}{\mathbb{E}_n[\mathcal{W}(\mathbf{x}_{[n]})]}\right].$$

The optimal values for $R_1, \dots, R_{N-1}$ that minimize the variance of the estimate $\log \widehat{Z}$ from (5) are

$$R_n^{\text{opt}} = \frac{\sqrt{v_n}}{\sum_{n'=1}^{N-1}\sqrt{v_{n'}}} R_{\text{tot}}\,. \tag{6}$$

The corresponding minimum variance of $\log \widehat{Z}$ is

$$V_{\min} = \big(\textstyle\sum_{n=1}^{N-1}\sqrt{v_n}\big)^2 / R_{\text{tot}}\,. \tag{7}$$

Proof. Due to the independence of the samples, the variance that we want to minimize can be decomposed as

$$\mathbb{E}_1 \dots \mathbb{E}_{N-1}\left[(\log\widehat{Z} - \log Z)^2\right] = \sum_{n=1}^{N-1} \mathbb{V}_n\left[\log m_n\right],$$

where $m_n = \frac{1}{R_n}\sum_{r=1}^{R_n}\mathcal{W}(\mathbf{x}_{[n]}^r)$ represents a sample average. For large R_n, the central limit theorem implies that m_n converges in distribution to a Gaussian $\mathcal{N}(\mathbb{E}_n[\mathcal{W}(\mathbf{x}_{[n]})], \frac{1}{R_n}\mathbb{V}_n[\mathcal{W}(\mathbf{x}_{[n]})])$, hence the delta method can be used to approximate $\log m_n$:

$$\log m_n \approx \mathcal{N}\left(\log \mathbb{E}_n[\mathcal{W}(\mathbf{x}_{[n]})],\ \frac{v_n}{R_n}\right).$$

Having found an expression for $\mathbb{V}_n[\log m_n]$, the constrained optimization problem can therefore be rewritten as

$$\min_{R_1,\dots R_{N-1}} \sum_{n=1}^{N-1}\frac{v_n}{R_n} \quad \text{subject to} \quad \sum_{n=1}^{N-1} R_n = R_{\text{tot}}\,,$$

and can be solved using Lagrange multipliers to find, after some calculations, the results stated in the theorem. □

Theorem 1 implies that, given a fixed computational budget, a variable number of particles has to be used per iteration if the variance of $\log \widehat{Z}$ is to be minimized. From (6) we deduce that, as expected, when the variance v_n of the normalized weight updates is big, i.e. if $p_n(\mathbf{x}_{[n]})$ and $p_{n+1}(\mathbf{x}_{[n]})$ are not similar enough, then a higher number of particles is required.

Adaptively adding particles per iteration. Assuming that the computational budget can be exceeded in iteration n, how can particles be added adaptively so that $V_{\min}$ is decreased? We can glean some insight by considering the contribution of iteration n to $V_{\min}$. Firstly, we obtain an approximation to v_n with its empirical estimate

$$\widehat{v_n} = \frac{\frac{1}{R_n}\sum_r(\mathcal{W}(\mathbf{x}_{[n]}^r) - m_n)^2}{m_n^2} = \frac{R_n}{R_{\text{eff},n}} - 1\,,$$

[1] Asymptotically, residual resampling will always outperform multinomial resampling [10].

where $R_{\text{eff},n} \doteq R_{\text{eff}}([w_{n+1}^{[R_n]}])$. Assuming that $R_i = R_i^{\text{opt}}$ for all iterations, the contribution of iteration n can be isolated in $V_{\min}$ by substituting all $\widehat{v_i}$ into (7):

$$V_{\min} \approx \frac{\left(\sqrt{R_n^{\text{opt}}/R_{\text{eff},n}^{\text{opt}} - 1} + \sum_{i\neq n}\sqrt{R_i^{\text{opt}}/R_{\text{eff},i}^{\text{opt}} - 1}\right)^2}{R_n^{\text{opt}} + \sum_{i\neq n} R_i^{\text{opt}}} . \quad (8)$$

If we are now allowed to exceed the computational budget, we can visualize one possible way to further decrease the variance $V_{\min}$. If at iteration n we increased the number of particles to $R_n^* > R_n^{\text{opt}}$ particles that still target $p_n(\mathbf{x}_{[n]})$, then $V_{\min}$ could be decreased provided that $R_{\text{eff},n}^{\text{opt}}/R_n^{\text{opt}} < R_{\text{eff},n}^*/R_n^* \leq 1$. The variance decreases when particles are added so that the ESS *per particle* increases.[2] This intuition represents the starting point for the sampler that is developed next.

3. Adaptive Resample-Move

Theorem 1 dictates how to optimally divide a fixed particle budget if the variances v_n of $w_{n+1}/\mathbb{E}_n[w_{n+1}]$ are known under i.i.d. conditions for $n = 1, \ldots, N-1$. With Algorithm 1 being sequential and having no knowledge of future iterations $n' > n$, we can greedily try to keep R_{tot} small, by using the ESS as a gauge for adaptively setting R_n. At a high level, iteration n starts with $R_n = R$ particles, and while a condition based on the ESS is not met, the iteration's number of particles is increased to $R_n := R_n + R$ through various means, as explained in Section 3.2 (at most $i_{\max}$ times). This ensures $\frac{1}{N-1}R_{\text{tot}} \leq i_{\max}R$, although experimentally the average number of particles is much smaller, with $\frac{1}{N-1}R_{\text{tot}} \approx 1.5R$ for Section 4's results. We introduce this adaptive "generate" loop to the RM in Algorithm 2, and call it *Adaptive Resample-Move* (ARM). Note that only few lines in Algorithm 1 need to be changed. Whenever a better approximation of the probability distribution is needed, ARM generates new particles that target it. At the other extreme end, if all components of $\mathbf{x}$ are independent, it captures the fact that no application of a move-step transition kernel would ever be required.

We next present two sufficient conditions that allow us to enlarge the particle set at any iteration n.

Proposition 1. *Let $(\chi_{[n]}^{[R_n]}, \widetilde{\rho}_n^{[R_n]})$ be a set of R_n particles that target $p_n(\mathbf{x}_{[n]})$. If we have R new particles $(\mathbf{x}_{[n]}^{[R]}, \widetilde{w}_n^{[R]})$ such that*

1. *$(\mathbf{x}_{[n]}^{[R]}, \widetilde{w}_n^{[R]})$ targets $p_n(\mathbf{x}_{[n]})$, and*
2. *the $R_n + R$ particles' weights are rescaled as $\alpha\widetilde{\rho}_n^{[R_n]}$ and $(1-\alpha)\widetilde{w}_n^{[R]}$, with $0 \leq \alpha \leq 1$,*

then $((\chi_{[n]}^{[R_n]}, \mathbf{x}_{[n]}^{[R]}), (\alpha\widetilde{\rho}_n^{[R_n]}, (1-\alpha)\widetilde{w}_n^{[R]}))$ also targets $p_n(\mathbf{x}_{[n]})$. This also holds if the new set $\mathbf{x}_{[n]}^{[R]}$ is moved using a transition kernel that leaves $p_n(\mathbf{x}_{[n]})$ invariant.

[2] Looking at the ESS alone is not sufficient. As a simple argument, the ESS in (4) can be doubled by simply duplicating each particle, but this doesn't alter the ESS per particle.

Algorithm 2 Adaptive Resample-Move

```
   ⋮
3: w_{n+1}^{[R]} := W(x_{[n]}^{[R]}) w̃_n^{[R]}                    // smooth
   if γ^(0) < γ_thr then
      χ_{[n]}^{[R]} := ∅;  ρ̃_n^{[R]} := ∅;  i := 0
   end if
   while γ^(i) < γ_thr and i < i_max do                          // generate
      x_{[n]}^{[R]} ~ K(x_{[n]} ; x_{[n]}^{[R]})                 // move
      χ_{[n]}^{[R_n]} := (χ_{[n]}^{[R_n]}, x_{[n]}^{[R]})        // include samples
      ρ̃_n^{[R_n]} := (R_n/(R_n+R) ρ̃_n^{[R_n]}, R/(R_n+R) w̃_n^{[R]})
      w_{n+1}^{[R_n]} := W(χ_{[n]}^{[R_n]}) ρ̃_n^{[R_n]}          // smooth
      i := i + 1
   end while
   ⋮
7: if γ^(i) < γ_thr or R_n > R then
8:    x_{[n]}^{[R]} := resample(w̃_{n+1}^{[R_n]}, χ_{[n]}^{[R_n]})   // resample
   ⋮
```

Proof. As $(\chi_{[n]}^{[R_n]}, \widetilde{\rho}_n^{[R_n]})$ and $(\mathbf{x}_{[n]}^{[R]}, \widetilde{w}_n^{[R]})$ both target $p_n(\mathbf{x}_{[n]})$, we have

$$\alpha \textstyle\sum_{r=1}^{R_n} \varphi(\chi_n^r)\widetilde{\rho}_n^r \longrightarrow \alpha\, \mathbb{E}_n[\varphi(\mathbf{x}_{[n]})]$$
$$(1-\alpha) \textstyle\sum_{r=1}^{R} \varphi(\mathbf{x}_n^r)\widetilde{w}_n^r \longrightarrow (1-\alpha)\mathbb{E}_n[\varphi(\mathbf{x}_{[n]})] .$$

The first statement is verified by combining the two sums. It is also possible to move the new particles $\mathbf{x}_{[n]}^{[R]}$ using a transition kernel that leaves $p_n(\mathbf{x}_{[n]})$ invariant, as they would still target $p_n(\mathbf{x}_{[n]})$; see [3] for details. □

3.1. Adding a generate-loop

We exploit Proposition 1 to expand the move-step into a generate-loop in Algorithm 2, by starting with a base level of $R_n^{(0)}$ particles that is large enough to get a reliable estimate of the effective sample size $R_{\text{eff},n}^{(0)}$. Defining

$$\gamma^{(i)} = R_{\text{eff},n}^{(i)}/R_n^{(i)} ,$$

for generate-iteration i, we see from (8) that $R_{\text{eff},n}^{(0)}$ would contribute approximately $\sqrt{1/\gamma^{(0)} - 1}$ to the minimum variance (via the first square root term). If $\gamma^{(0)}$ is below a threshold value γ_{thr}, and more particles are generated such that the ratio $\gamma^{(1)}$ is increased, then, as suggested in Section 2.3, the variance could be further decreased. This procedure is iterated until $\gamma^{(i)} > \gamma_{\text{thr}}$, or until a maximum number of iterations $i_{\max}$ are reached. The threshold should be as close as possible to one, as the contribution to the variance in (8) is $\sqrt{1/\gamma_{\text{thr}} - 1}$. One should however bear in mind that a higher threshold gives a computationally more expensive algorithm. In our experiments, a value of $\gamma_{\text{thr}} = 0.7$ gave a good trade-off.

As a cautionary tale, the ESS may be misleading, as it could be high even if an important mode in the distribution is missed. However, it gives a practically useful measurement of the quality of the approximation, and is hence commonly used in SMC

methods [8]. Additionally, Theorem 1 rested on an assumption of independent, equally-weighted samples to make the study of some asymptotic properties of the introduced sampler feasible. In practice, the resample-step introduces correlations, and the particles may not yet be at equilibrium after the application of $\mathcal{K}$. However, as our results in the following sections suggest, ARM allows us to reduce the variance of the estimate of normalizing constants, even if these assumptions are not fully satisfied.

3.2. Generating the particles

As long as Proposition 1's properties are satisfied, any method could be used to generate new particles. As shown in Algorithm 2, ARM creates R new particles by repeating and moving the old set $(\mathbf{x}_{[n]}^{[R]}, \widetilde{w}_n^{[R]})$ to automatically target $p_n(\mathbf{x}_{[n]})$. In Section 5 we experimented with two alternative variations of ARM. The variations are different mechanisms to generate a new set targeting $p_n(\mathbf{x}_{[n]})$:

ARM-anticipate. As done in ARM, this method starts by copying the old set of particles, $(\mathbf{x}_{[n]}^{[R]}, \widetilde{w}_n^{[R]})$. Before moving this new set, more copies of the particles are made, so that $p_{n+1}(\mathbf{x}_{[n]})$ is better approximated. This anticipates the information given by p_{n+1}. Borrowing an idea from residual resampling [9], these more promising particles can be, for instance, those whose indexes are in $\mathcal{L} \doteq \{r : R\lfloor \widetilde{w}_{n+1}^r \rfloor > 0\}$. Particles with indexes in $\mathcal{L}$ can be split into $N_r = \lfloor R\widetilde{w}_{n+1}^r \rfloor$ copies, and their weights set to $\widetilde{w}_r^n / N_r$ to ensure that the new set still targets $p_n(\mathbf{x}_{[n]})$. Particles with indexes not in $\mathcal{L}$ are kept as they are. A transition kernel that leaves $p_n(\mathbf{x}_{[n]})$ invariant is then applied to them. The total number of created particles will be $S = R + \sum_r N_r$.

ARM-reseed. New particles $\mathbf{x}_{[n]}^{[S]} \sim p_n(\mathbf{x}_{[n]})$ may be generated from any other sampler, like an MCMC algorithm, weighted with $\widetilde{w}_n^{[S]} = 1/S$, and added to the old set of particles. The motivation is that the newly added particles are completely independent from the current set, and may therefore be from high density regions in p_n that were approximated poorly before, possibly allowing a significant increase in $\gamma^{(i)}$. This method is entirely application-specific, and running the new sampler to convergence to obtain $\mathbf{x}_{[n]}^{[S]}$ might be a costly operation.

4. Gaussian Process Classification

As a first evaluation of ARM, we consider a Gaussian Process (GP) classification model (GPC). A GP specifies a prior distribution on functions $x : \boldsymbol{\xi} \to \mathbb{R}$, so that its values $x_n = x(\boldsymbol{\xi}_n)$ are correlated through a prior covariance matrix $\mathbf{K}$ that depends on the inputs $\boldsymbol{\xi}_n$. In a GPC model an observed class label $y_n \in \{-1, +1\}$ depends on x_n through a likelihood $p(y_n|x_n) = \Phi(y_n x_n)$, the probit link function being $\Phi(x) = \int \Theta(z)\mathcal{N}(z; x, 1)\,\mathrm{d}z$. The step function $\Theta(z)$ is one if its argument is nonnegative, and zero otherwise. Using the step function, the joint model is

$$p(\mathbf{y}, \mathbf{z}, \mathbf{x}) = p(\mathbf{y}|\mathbf{z})\, p(\mathbf{z}|\mathbf{x})\, p(\mathbf{x}) = \prod_{n=1}^{N} \Theta(y_n z_n)\, \mathcal{N}(z_n; x_n, 1) \cdot \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{K}) ,$$

and we are interested in the marginal likelihood $Z = p(\mathbf{y})$ as a function of $\mathbf{K}$. Two representations of Z arise from either integrating out $\mathbf{z}$ to give

$$p(\mathbf{x}|\mathbf{y}) = \frac{1}{p(\mathbf{y})} \exp\left(-\frac{1}{2}\mathbf{x}^T \mathbf{K}^{-1}\mathbf{x} + \sum_n \log \Phi(y_n x_n) + c_1 \right) \quad (9)$$

with $c_1 = \frac{N}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{K}|$, or integrating out $\mathbf{x}$ to yield

$$p(\mathbf{z}|\mathbf{y}) = \frac{1}{p(\mathbf{y})} \exp\left(-\frac{1}{2}\mathbf{z}^T (\mathbf{K} + \mathbf{I})^{-1}\mathbf{z} + \sum_n \log \Theta(y_n z_n) + c_2 \right) , \quad (10)$$

where $c_2 = \frac{N}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{K} + \mathbf{I}|$. Uncertainty is shifted from the likelihood to the prior between these two representations.

4.1. Implementation

ARM can be implemented using either the formulation in (9) or in (10). There are subtle differences between their MCMC transition kernels in the move-step. The kernel could be a Gibbs sampler for each variable in $p(x_i|y_i, \mathbf{x}_{[n]\setminus i})$ or in $p(z_i|y_i, \mathbf{z}_{[n]\setminus i})$, where \ is read as "without". The Gibbs sweeps for $i = 1, \ldots, n$ are parameter-free, and for the number of particles under consideration, computationally much faster than kernels that make use of gradient information.

The representation in (10) allows for a more efficient sampler in the move-step than (9) (see Appendix A). The parameter-free Gibbs sweep has variances either $1/[\mathbf{K}^{-1}]_{ii}$ for (9), or $1/[(\mathbf{K}+\mathbf{I})^{-1}]_{ii}$ in the case of (10). If we introduce the scaling of the covariance function in $\mathbf{K} = \sigma^2\mathbf{K}_0$, it is easy to see that the variance of the Gibbs sampler scales with σ^2 for both formulations when $\sigma^2 \gg 1$, but for $\sigma^2 \ll 1$ the variance scales with σ^2 for (9) and is constant for (10). Gibbs sampling from (10) is thus more widely applicable as the step-size will in general be larger. Furthermore, the representation (10) has the additional advantage as it is amenable to fast *slice sampling* and avoids the computation of inverse Gaussian cumulative density functions Φ^{-1}. The details of all the steps necessary to implement ARM for GPC are given in Appendix A.

4.2. Experimental results

We evaluate the efficiency of ARM on the USPS 3-vs.-5 data set [12], using a covariance function $K_{mn} = k(\boldsymbol{\xi}_m, \boldsymbol{\xi}_n) = \sigma^2 \exp(-\frac{1}{2}\|\boldsymbol{\xi}_m - \boldsymbol{\xi}_n\|^2/\ell^2)$ that correlates inputs $\boldsymbol{\xi}_m$ and $\boldsymbol{\xi}_n$ through a length scale $\ell = \exp(4.85)$ and amplitude parameter $\sigma = \exp(5.1)$.[3] Our main basis for comparison is Annealed Importance Sampling (AIS) [4] using different versions

[3] ARM was evaluated on [12]'s entire $(\log\ell, \log\sigma)$-grid, of which this setting proved to be the hardest.

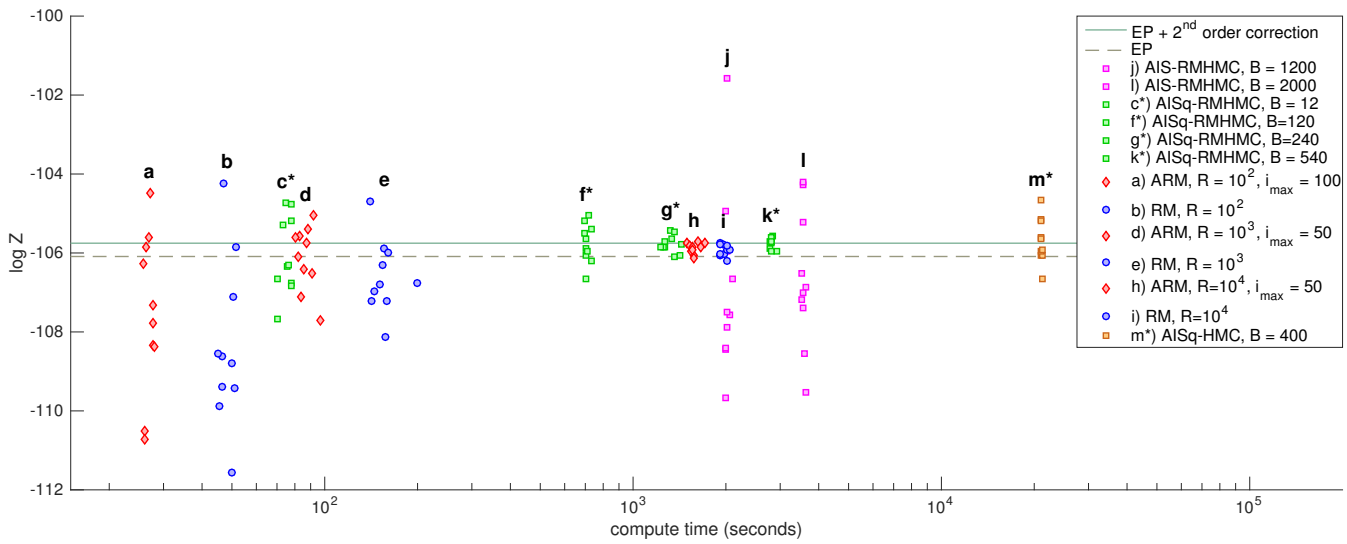

Figure 2: An extensive time-based comparison for GPC on the USPS 3-vs.-5 data set, using a highly correlated prior. From left to right, the evaluations are for (a) ARM with $R = 10^2, i_{\max} = 100$; (b) RM with $R = 10^2$; (c*) AISq-RMHMC using $B = 12$; (d) ARM with $R = 10^3, i_{\max} = 50$; (e) RM with $R = 10^3$; (f*) AISq-RMHMC using $B = 120$; (g*) AISq-RMHMC using $B = 240$; (h) ARM with $R = 10^4, i_{\max} = 50$; (i) RM with $R = 10^4$; (j) AIS-RMHMC using $B = 1200$; (k*) AISq-RMHMC using $B = 540$; (l) AIS-RMHMC using $B = 2000$; (m*) AISq-HMC. The dotted line indicates EP's approximation, and the solid line a second order correction to the EP solution. A label that is starred indicates that an AIS method was aided by annealing from EP's $q(\mathbf{x})$ to $p(\mathbf{x}|\mathbf{y})$, and not from the prior, as is more commonly done.

of Hamiltonian Monte Carlo (HMC) methods for the transition kernel.[4] Such a highly correlated high-dimensional prior highlights some deficiencies in a basic HMC method, where mixing can be slow due to a sample's leapfrog trajectory oscillating up and down the sides of a valley of $\log p(\mathbf{y}|\mathbf{x})^\beta p(\mathbf{x})$, without actually progressing through it. In order to get a working HMC sampler for the problem, we derived a Riemannian Manifold HMC method (RMHMC) [5] for GP models. To our knowledge, this has not been done before, and as it would be of independent interest, detailed pseudo-code is given in [6]. To further aid AIS with different HMC methods, we additionally let AIS anneal from a Gaussian approximation $q(\mathbf{x})$ to the GPC posterior, instead of the prior. The approximation $q(\mathbf{x})$ was obtained with Expectation Propagation (EP).

To test the importance of the the MCMC kernel, we also compared the proposed method against a more basic SMC algorithm with no move-step, using a high number of particles that matched the computational budget of ARM. However, the estimates obtained with this sampler were much worse than the ones obtained with ARM, and are therefore not included in the following analysis.

Figure 2 compares the estimates of $\log Z$ obtained with ARM and AIS to the required computation time. Broadly, we see that ARM makes better use of a computational budget than RM. Secondly, as the *only* competitive versions of AIS with HMC have to rely on outside information through $q(\mathbf{x})$, SMC methods, in the spirit of "hot coupling" [13], are unequivocally better workhorses for estimating normalizing constants in this context. The details of the methods are:

ARM in (a), (d) and (h) uses $i_{\max} = 50$ (100 for (a)), with $R_n^{(0)} = R = 10^2, 10^3, 10^4$ respectively. Residual resampling is done when $R_{\text{eff}} < 0.9R$.

RM in (b), (e) and (i) uses two Gibbs sweeps in each move-step, with $R = 10^2, 10^3, 10^4$, and does residual resampling only when $R_{\text{eff}} < 0.9R$. The choice of two sweeps is made to approximately match the variance of ARM in the estimate of the normalizing constant, and it is empirically equivalent to using more particles with only 1 sweep.

AISq+HMC in (m*) runs AIS from the EP's $q(\mathbf{x})$ at $\beta = 0$ to $p(\mathbf{x}|\mathbf{y})$ at $\beta = 1$ using intermediate distributions

$$p_\beta(\mathbf{x}) = \frac{1}{Z(\beta)} \left(\prod_n \Phi(y_n x_n) \cdot \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{K}) \right)^\beta q(\mathbf{x})^{1-\beta} . \quad (11)$$

Note that a starred label indicates that the estimates were aided by $q(\mathbf{x})$. A HMC transition kernel with $l_{\max} = 200$ *leapfrog* steps is used at each $\beta \in [0, 1]$ value. AIS's β-grid is a geometric progression over $B = 400$ β-values. Plot (m*) used a step size $\epsilon = 0.02$ per proposal; both $l_{\max}$ and ϵ were carefully tuned to the problem. The simplest AIS-HMC version, which anneals from $p(\mathbf{x})$ and not $q(\mathbf{x})$, didn't obtain estimates inside the bounds of Figure 2, and is excluded.

AIS+RMHMC in (j) and (l) anneals from $p(\mathbf{x})$, and replaces HMC with a more advanced RMHMC that uses $\epsilon = 0.1$ and $l_{\max} = 10$ leapfrog steps per proposal at each β value.

[4] The posterior density is very correlated. The mixing rate of a single Gibbs sampler was too slow in our simulations to get a competitive estimate of the normalizing constant, when used in conjunction with AIS.

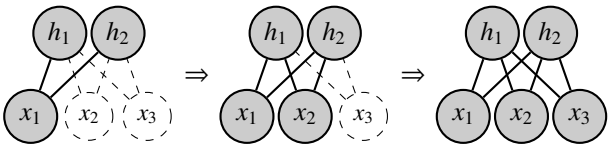

Figure 3: Sequential formation of a simple RBM.

AISq+RMHMC in (c*), (f*), (g*) and (k*) anneals from $q(\mathbf{x})$ using a RMHMC kernel ($\epsilon = 0.1, l_{\max} = 10$).

It is known that the EP estimate of $\log Z$ is remarkably accurate for this problem [12], hence EP's $\log Z$ estimate and its a second-order corrected estimate [14] are given for reference.

Our last observation is a practical one. ARM and RM are simple and tend to be more robust than AIS with HMC or Metropolis-Hastings, as they have little dependence on external parameters. HMC, on the other hand, relies on carefully tuned settings of ϵ and $l_{\max}$, or requires more complicated extensions like RMHMC used here, or approaches like the No-U-Turn sampler [15].

5. Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a bipartite binary graphical model, connecting visible units $\mathbf{x} \in \{0,1\}^N$ to hidden binary units $\mathbf{h} \in \{0,1\}^H$ through

$$p(\mathbf{x},\mathbf{h}) = \frac{f(\mathbf{x},\mathbf{h})}{Z} = \frac{1}{Z}\exp(\mathbf{x}^T\mathbf{W}\mathbf{h} + \mathbf{a}^T\mathbf{x} + \mathbf{b}^T\mathbf{h}) \,.$$

The $N \times H$ weight matrix $\mathbf{W}$ defines the connections between the two layers, $\mathbf{a}$ is a $N \times 1$ bias term relative to the visible units, and $\mathbf{b}$ is a $H \times 1$ bias term for the hidden units.

To sequentially form an RBM, we can start with a graph containing only the hidden units, and keep adding a new visible unit (with corresponding weights and bias) at each iteration; see Figure 3. Instead of working with the joint distribution of visible and hidden units, it is more convenient to have the latter summed out:

$$p_n(\mathbf{x}_{[n]}) = \frac{1}{Z_n} e^{\mathbf{a}_{[n]}^T \mathbf{x}_{[n]}} \prod_{h=1}^{H} \left(1 + e^{b_h + \mathbf{W}_{[n],h}^T \mathbf{x}_{[n]}}\right) \,. \tag{12}$$

The initialization of the ARM algorithm is straight-forward, as it is easy to sample from $p_{[1]}(x_{[1]}; a_{[1]}, \mathbf{b}, \mathbf{W}_{[1],[H]})$ and compute Z_1. In the move-step, a possible parameter-free transition kernel is the standard Gibbs sampler in which first we sample $\mathbf{h}|\mathbf{x}$ and then $\mathbf{x}|\mathbf{h}$ in parallel [16]. To improve mixing, one may repeatedly apply the transition kernel (we used 10 iterations in our experiments). The weight updates used in the smooth-step are

$$\mathcal{W}\left(\mathbf{x}_{[n]}^r\right) = \sum_{x_{n+1}} e^{a_{n+1}x_{n+1}} \prod_{h=1}^{H} \frac{1 + e^{g_{n,h}^r + W_{n+1,h}x_{n+1}}}{1 + e^{g_{n,h}^r}} \,, \tag{13}$$

where $g_{n,h}^r \doteq b_h + \mathbf{W}_{[n],h}^T \mathbf{x}_{[n]}^r$. The two terms in the sum in (13), normalized by $\mathcal{W}(\mathbf{x}_{[n]}^r)$, form $p(x_{n+1}|\mathbf{x}_{[n]}^r)$ in the augmentation-step.

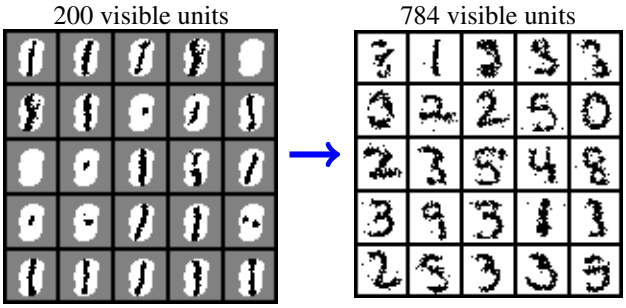

Figure 5: Example of particles approximating $p_n(\mathbf{x}_{[n]})$ using the RBM model trained with PCD. Black and white pixels represent ones and zeros, while excluded visible units are colored gray.

5.1. Experimental results

We compare the performance of ARM on the two most difficult RBM models used in [17]. Both models were trained on the MNIST handwritten digits dataset [18], the first one with persistent contrastive divergence (PCD) [19] and the second one with contrastive divergence [16] with 25 steps of Gibbs sampling (CD25). The RBMs have 784 visible units and 500 hidden units, making the exact computation of Z intractable. In [17] the partition function is estimated with AIS, using a path based on averaging the moments of the initial and target distribution instead of the usual geometric one. The algorithm presented is computationally very expensive: first the moments of the target distributions are estimated using 10^3 independent Gibbs chains with 11000 Gibbs steps each, then the parameters of 9 intermediate RBMs have to be fit in order to match the averaged moments at 9 different temperatures (knots of a spline), and finally a geometric path with 10^4 intermediate distributions is used in order to pass from one RBM at one knot to the next one, therefore giving $K = 10^5$ intermediate distributions in total. The best performing initial distribution used for AIS in [17] is the base rate RBM, in which the visible biases are set to the average pixel values in the MNIST training set and all the other parameters are set to 0. In a similar way information on the training data can be exploited by ARM as well: units are introduced starting from the most active ones (i.e. those whose variance in the training set is higher) so that higher density regions are explored from the very beginning.

Figure 5 shows some of the particles that were generated by ARM at iteration $n = 200$ and $n = 784$. We see that the visible units (pixels of the image in this case) are sequentially added, and when n is high enough the particles start looking like real handwritten digits digits.

The particle set can be extended with S new independent particles $\mathbf{x}_{[n]}^{[S]}$ using ARM-reseed (see Section 3.2). To ensure that the new set of particles targets $p_n(\mathbf{x}_{[n]})$, a few thousand steps of the same Gibbs kernel of the move-step are applied to S examples sampled randomly from the MNIST training set. To minimize the computational overhead of this costly step, the actual number of new particles generated at iteration i was $S = \max\left\lfloor(1 - \gamma^{(i)}/\gamma_{\text{thr}})R, 100\right\rfloor$, where R is the baseline number of particles. This means that if the current set of particles is already a good approximation of the distribution of interest, only

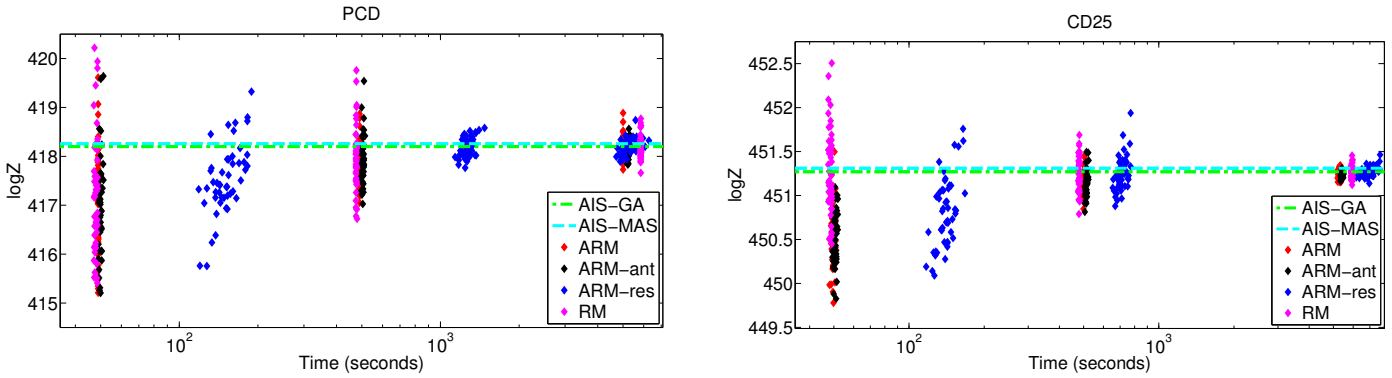

Figure 4: Comparison between ARM, RM and AIS for two different RBMs, labelled as PCD and CD25. *For both models*: in all the simulations we used $t = 10$ Gibbs steps for the transition kernel of the move-step, $\gamma_{\text{thr}} = 0.7$, ARM and ARM-anticipate used $R = 10^2, 10^3, 10^4$ particles and $i_{\max} = 3$, RM used $R = 10^2, 10^3, 1.2\times10^4$ particles. *For PCD*: ARM-reseed used $R = 10^2, 10^3, 6\times10^3$ particles, $i_{\max} = 2$ and 3000 Gibbs transitions when generating new particles. *For CD25*: ARM-reseed used $R = 10^2, 10^3, 10^4$ particles, $i_{\max} = 2$ and 1500 Gibbs transitions when generating new particles. The results for AIS were extracted from [17], and were obtained with 5000 chains, 10^5 intermediate distributions and 1 Gibbs transition for each intermediate distribution.

few new particles are created. At least 100 particles were generated at each step, so that the new set provides a sufficiently good approximation to $p_n(\mathbf{x}_{[n]})$.

The results from 50 runs of ARM and its two variants from Section 3.2 are shown in Figure 4, using an increasing number of particles. As a reference we also show the results for AIS from [17], which were obtained with a geometric averages (AIS-GA) path and a moment averages spline (AIS-MAS) path, using $K = 10^5$ intermediate distributions.

Remarkably, ARM allows us to get very close to the results obtained with AIS in less than a minute of computation time. Most importantly, for ARM the tuning of the parameters was almost effortless, as these are merely a function of the allowed time budget. By considering the estimates in Figure 4 that were obtained with the highest number of particles (the rightmost estimates in each plot), we notice that ARM reduces the variance of RM estimates using less computational power. ARM-anticipate, which creates new particles in a "smarter" way, outperforms simple ARM in terms of variance of the estimates. For the RBM trained with PCD, generating a new set of independent particles with ARM-reseed significantly improves the efficiency of the sampler; see in particular the results obtained with a baseline of $R = 10^3$. On the other hand, other methods give comparable results to ARM-reseed in less time for the CD25-trained RBM. As noted in [17], PCD seems to be a more difficult model to sample from, and as such a completely new set of particles could be beneficial.

We can get an insight into the computational performance of both ARM and AIS by comparing the efficiency of the respective Gibbs kernels, as they are by far the most time consuming operation in both algorithms. One particle of ARM essentially corresponds to one run of AIS. At iteration n of ARM, the Gibbs sampler has complexity $O(tHn)$, where t is the number of Gibbs steps used. As $n = 1, \ldots, N$, the overall complexity is $O(tHN^2)$. The complexity of AIS (using geometric or moment averages), given K intermediate distributions, is $O(tHNK)$. We than see that AIS is much more computationally expensive than ARM, as it typically requires $K \gg N$ to get accurate estimates of normalizing constants (our best performing setups in Figure 4 use $K = 10^5$ and $N = 784$).

6. Related Work

To estimate normalizing constants in smaller scale models, stochastic approximation techniques are the first choice, as they can lead to very accurate results given enough computational time. AIS [4], used as a comparison in our simulations, is one of the most widely used method for estimating normalizing constants. It belongs to a more general family of methods, known as *tempering methods*, that are based on a one-parameter $\beta \in [0, 1]$ extension of the model: $p_\beta(\mathbf{x}) = f(\mathbf{x}, \beta)/Z(\beta)$ such that we interpolate between a usually tractable $p_0(\mathbf{x})$ and the model of interest $p(\mathbf{x}) = p_1(\mathbf{x})$. This was done in (11). The normalizer can then be written as an integral,

$$\log Z - \log Z(0) = \int_0^1 \mathbb{E}_{p_\beta}\left[\frac{\mathrm{d}}{\mathrm{d}\beta}\log f(\mathbf{x}, \beta)\right]\mathrm{d}\beta\,, \qquad (14)$$

that in practice will have to be discretized. AIS provides in general accurate estimates, but relies on often difficult hand tuning and a high number of intermediate distributions to limit the large variances of the estimate introduced by the discretization of the continuous temperature scale. A theoretical derivation of this statement can be found in the Appendix B. For standard annealing schemes, such as AIS, the analysis shows that the spacing between βs should be roughly $1/\sqrt{V(\beta)}$ with

$$V(\beta) \doteq \mathbb{V}_{p_\beta}\left[\frac{\mathrm{d}\log f(\mathbf{x}, \beta)}{\mathrm{d}\beta}\right].$$

The averaging moments annealing algorithm [17] may be viewed as a scheme for setting the interpolating distributions for exponential families in a way less prone to discretization errors, using moment averages to define the sequence of consecutive distributions rather than the standard choice of geometric averages [20].

Sequential Monte Carlo (SMC) algorithms can take an alternative route to constructing an interpolation scheme to estimate Z. Our choice of sequence is motivated by a computationally efficient implementation of versions of Hamze and de Freitas's [13] "hot coupling" samplers (see Algorithms 1 and 2), as well a discrete decomposition for $\log Z$ that allows one to estimate it accurately. With methods such as the ones introduced in [13] or ARM, a discrete decomposition for the normalizing constant naturally arises; see (2). In the results in Section 4 and 5 these methods showed superior performance to AIS, but a more general statement is not possible. AIS may be a better choice for other models, despite being more difficult to tune than ARM.

7. Conclusion

In this paper we introduced *Adaptive Resample-Move*, an SMC algorithm that reduces the variance of estimates of normalizing constants by expanding the particle set whenever a better approximation of an intermediate distribution is needed. A theoretical justification for ARM is also given under ideal conditions. Experimental results on two challenging models previously analyzed by other authors (GPC and RBMs), show that despite its simplicity and the minimal tuning required, ARM allows to efficiently find accurate estimates of normalizing constants, and should therefore be considered as a valid alternative to AIS.

Acknowledgements

Marco Fraccaro is supported by Microsoft Research through its PhD Scholarship Programme.

References

[1] B. Paige, F. Wood, A. Doucet, Y. W. Teh, Asynchronous anytime sequential monte carlo, in: Advances in Neural Information Processing Systems, 2014.

[2] N. Whiteley, A. Lee, K. Heine, On the role of interaction in sequential monte carlo algorithms, arXiv:1309.2918 [stat.CO].

[3] W. R. Gilks, C. Berzuini, Following a moving target – Monte Carlo inference for dynamic Bayesian models, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63 (1) (2001) 127–146.

[4] R. M. Neal, Annealed importance sampling, Statistics and Computing 11 (2) (2001) 125–139.

[5] M. Girolami, B. Calderhead, Riemann manifold Langevin and Hamiltonian Monte Carlo methods, Journal of the Royal Statistical Society, Series B 73 (2) (2011) 123–214.

[6] U. Paquet, M. Fraccaro, An efficient implementation of Riemannian manifold Hamiltonian Monte Carlo for Gaussian process models, www.ulrichpaquet.com/rmhmc.pdf.

[7] A. Doucet, N. de Freitas, N. Gordon, An introduction to sequential Monte Carlo methods, in: Sequential Monte Carlo Methods in Practice, Statistics for Engineering and Information Science, 2001, pp. 3–14.

[8] P. Del Moral, A. Doucet, A. Jasra, Sequential Monte Carlo samplers, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68 (3) (2006) 411–436.

[9] J. S. Liu, R. Chen, Sequential Monte Carlo methods for dynamic systems, Journal of the American Statistical Association 93 (443) (1998) 1032–1044.

[10] N. Chopin, Central limit theorem for sequential monte carlo methods and its application to Bayesian inference, Ann. Statist. 32 (6) (2004) 2385–2411.

[11] N. Chopin, A sequential particle filter method for static models, Biometrika 89 (3) (2002) 539–552.

[12] M. Kuss, C. E. Rasmussen, Assessing approximate inference for binary Gaussian process classification, Journal of Machine Learning Research 6 (2005) 1679–1704.

[13] F. Hamze, N. de Freitas, Hot coupling: A particle approach to inference and normalization on pairwise undirected graphs of arbitrary topology, in: Advances in Neural Information Processing Systems 18, 2005.

[14] M. Opper, U. Paquet, O. Winther, Perturbative corrections for approximate inference in Gaussian latent variable models, Journal of Machine Learning Research 14 (Sep) (2013) 2857–2898.

[15] M. D. Hoffman, A. Gelman, The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo, Journal of Machine Learning Research 15 (Apr) (2014) 1593–1623.

[16] G. E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Compututation 14 (8) (2002) 1771–1800.

[17] R. B. Grosse, C. J. Maddison, R. Salakhutdinov, Annealing between distributions by averaging moments, in: Advances in Neural Information Processing Systems 26, 2013, pp. 2769–2777.

[18] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proceedings of the IEEE, 1998, pp. 2278–2324.

[19] T. Tieleman, Training restricted Boltzmann machines using approximations to the likelihood gradient, in: Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 1064–1071.

[20] R. Salakhutdinov, I. Murray, On the quantitative analysis of Deep Belief Networks, in: Proceedings of the 25th Annual International Conference on Machine Learning, 2008, pp. 872–879.

[21] A. Gelman, X. Meng, Simulating normalizing constants: From importance sampling to bridge sampling to path sampling, Statistical science (1998) 163–185.

Appendix A. Resample-Move for Gaussian Process classification

In this appendix, we present a MCMC transition kernel for Algorithm 1's move step, as done in iteration n. The transition kernel performs Gibbs sampling, where each step is performs numerically fast slice sampling. Efficient smooth and augmentation steps are also given. We repeat (10) here for $p(\mathbf{z}_{[n]}) = f_n(\mathbf{z}_{[n]})/Z_n$:

$$p(\mathbf{z}_{[n]}|\mathbf{y}_{[n]}) = \frac{1}{Z_n}\exp\Bigg(-\frac{1}{2}\mathbf{z}_{[n]}^T(\mathbf{K}_{[n]}+\mathbf{I}_{[n]})^{-1}\mathbf{z}_{[n]} + \sum_{i\in[n]}\log\Theta(y_i z_i) + c\Bigg),$$

where $c = \frac{n}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{K}_{[n]}+\mathbf{I}_{[n]}|$.

Appendix A.1. The move-step

The move-step in Section 4 requires an MCMC transition kernel to resample $\mathbf{z}_{[n]}^{[R]}$ given its current state. The move-step draws samples from $p(z_i|y_i, \mathbf{z}_{[n]\setminus i})$ for $i \in [n]$ in random order. Let $\mathbf{S}^{(n)} = \mathbf{\Sigma}_{[n]}^{-1} = (\mathbf{K}_{[n]}+\mathbf{I}_{[n]})^{-1}$ be the inverse prior covariance of $\mathcal{N}(\mathbf{z}_{[n]}; \mathbf{0}, \mathbf{\Sigma}_{[n]})$. The conditional distribution for any z_i is

$$p(z_i|y_i, \mathbf{z}_{[n]\setminus i}) \propto \Theta(y_i z_i)\,\mathcal{N}\left(z_i\,;\,\mu_i,\,[S_{ii}^{(n)}]^{-1}\right), \tag{A.1}$$

where the Gaussian has mean

$$\mu_i = -\sum_{j\in[n]\setminus i} S_{ij}^{(n)} z_j \,/\, S_{ii}^{(n)}.$$

A single Gibbs sweep requires an inner product for μ_i for each of n samples with (A.1), giving complexity $O(n^2)$ for $\mathcal{K}$. This inner loop dominates Algorithm 1's cost.

Appendix A.2. Slice sampling in the move-step

Each Gibbs sample from $p(z_i|y_i, \mathbf{z}_{[n]\setminus i})$ in (A.1) can be extremely efficiently drawn using a *slice sampler* that only requires two uniform random numbers and the computation of a square root. We start by drawing height $u \sim u|z_i, y_i$ uniformly between zero and $\exp(-S^{(n)}_{ii}(z_i-\mu_n)^2/2)$. The bounds of the slice are the two roots of the quadratic $S^{(n)}_{ii}(z_i-\mu_i)^2 + 2\log u = 0$, possibly clipped at zero according to the sign of y_i. These operations can be concatenated into four steps to update z_i

$$\begin{aligned}
1:&\quad v = ((z_i-\mu_i)^2 - 2[S^{(n)}_{ii}]^{-1}\log(\text{rand}))^{1/2}\\
2:&\quad b_{\text{lo}} = (\mu_i - v)\,\mathbb{I}[y_i < 0 \text{ or } \mu_i > v]\\
3:&\quad b_{\text{hi}} = (\mu_i + v)\,\mathbb{I}[y_i > 0 \text{ or } \mu_i < -v]\\
4:&\quad z_i = (b_{\text{hi}} - b_{\text{lo}})\text{rand} + b_{\text{lo}}\,,
\end{aligned} \tag{A.2}$$

where rand produces a $\mathcal{U}[0,1]$ sample, and $\mathbb{I}[\cdot]$ is one if its argument is true, and zero otherwise.

Appendix A.3. The smooth- and augmentation steps

The smooth-step, the conditional density for x_{n+1} for the augmentation-step, as well as the algorithm's next loop, require $\mathbf{S}^{(n+1)} = \mathbf{\Sigma}^{-1}_{[n+1]}$. We first expand the inverse with an $O(n^2)$ operation

$$\mathbf{S}^{(n+1)} = \begin{pmatrix} \mathbf{S}^{(n)} + \mathbf{s}\mathbf{s}^T/\varsigma & \mathbf{s} \\ \mathbf{s}^T & \varsigma \end{pmatrix},$$

using the block matrix invserion

$$\begin{aligned}
\varsigma &= \left(\Sigma_{n+1,n+1} - \mathbf{\Sigma}^T_{n+1,[n]}\mathbf{S}^{(n)}\mathbf{\Sigma}_{[n],n+1}\right)^{-1}\\
\mathbf{s} &= -\varsigma\mathbf{S}^{(n)}\mathbf{\Sigma}_{[n],n+1}
\end{aligned}.$$

Notation $\mathbf{\Sigma}_{[n],n+1}$ refers to the subvector in $\mathbf{\Sigma}$ that is indexed by rows $[n]$ and column $n+1$. On obtaining $\mathbf{S}^{(n+1)}$, the smoothing step calculates w^r by averaging the likelihood for y_{n+1} over a Gaussian conditional distribution $p(z_{n+1}|\mathbf{y}_{[n]}, \mathbf{z}_{[n]})$ with mean and variance

$$\begin{aligned}
m &\doteq -\frac{1}{S^{(n+1)}_{n+1,n+1}}\sum_{i=1}^{n} S^{(n+1)}_{i,n+1} z^r_i\\
\sigma^2 &\doteq \frac{1}{S^{(n+1)}_{n+1,n+1}}\,,
\end{aligned}$$

to yield

$$\begin{aligned}
w^r &= \int \Theta(y_{n+1}z_{n+1})\,\mathcal{N}(z_{n+1}; m, \sigma^2)\mathrm{d}z_{n+1}\\
&= \Phi(y_{n+1}\cdot m/\sigma)\,.
\end{aligned}$$

Appendix B. Tempering methods

Reference [21] derived an exact asymptotic expression for the bias due to the discretization of

$$\log Z = \log Z(1) - \log Z(0) = \int_0^1 \mathbb{E}_{p_\beta}\left[\frac{\mathrm{d}}{\mathrm{d}\beta}\log f(\mathbf{x},\beta)\right]\mathrm{d}\beta$$

in (14). This expression is not computable in practice and the main challenges of tempering methods are to come up with efficient procedures for choosing $f(\mathbf{x},\beta)$ [17, 21] and tuning the discretisation of β to the specific problem. Intuitively, a necessary requirement for the successful interpolation is that the intermediate distributions must be sufficiently similar. In other words, the distribution of the *energy* $\frac{\mathrm{d}\log f(\mathbf{x},\beta)}{\mathrm{d}\beta}$ for adjacent distributions must be overlapping. We define $M(\beta)$ as the expectation of the energy,

$$M(\beta) \doteq \mathbb{E}_{p_\beta}\left[\frac{\mathrm{d}}{\mathrm{d}\beta}\log f(\mathbf{x},\beta)\right],$$

and its change a $\Delta M(\beta) \doteq M(\beta+\Delta\beta) - M(\beta)$. $\Delta M(\beta)$ should be made of the same order as the fluctuations in energy, $\sqrt{V(\beta)}$ with $V(\beta) \doteq \mathbb{V}_{p_\beta}\left[\frac{\mathrm{d}\log f(\mathbf{x},\beta)}{\mathrm{d}\beta}\right]$. Combining therefore $\Delta M(\beta) \approx \sqrt{V(\beta)}$ with $\Delta M(\beta) \approx \frac{\mathrm{d}M(\beta)}{\mathrm{d}\beta}\Delta\beta$ we have a yardstick to measure how much we are allowed to change β:

$$\Delta\beta \approx \sqrt{V(\beta)}\Big/\frac{\mathrm{d}M(\beta)}{\mathrm{d}\beta}\,.$$

We can write $\frac{\mathrm{d}M(\beta)}{\mathrm{d}\beta} = U(\beta) + V(\beta)$ with $U(\beta) \doteq \mathbb{E}_{p_\beta}\left[\frac{\mathrm{d}^2\log f(\mathbf{x},\beta)}{\mathrm{d}\beta^2}\right]$. For standard tempering (as used in AIS) we have $\log f(\mathbf{x},\beta) = \beta\log f(\mathbf{x}) + (1-\beta)\log f_0(\mathbf{x})$ which gives $\Delta\beta \approx 1/\sqrt{V(\beta)}$. This result has the simple interpretation that if fluctuations are large we need to use a finer discretization, increasing therefore the computations required. Unfortunately, "phase transition" type behaviour, marked by a large increase in fluctuations for a specific β, may also occur in large statistical models.