An evaluation of Markov Chain Monte Carlo samplers for models with discrete parameters

Bernd van den Hoek

Master Thesis, ICA-5895391

Supervisors:
Dr. M.I.L. Vakar
T.J. Smeding, MSc
Dr. S. Renooij

Utrecht University

Utrecht University
Department of Information and Computing Sciences
July 22, 2022

Abstract

MCMC samplers are widely used in Bayesian inference. Samplers for models with continuous parameters are highly efficient and scalable. Creating algorithms for models with discrete parameters seems to be a lot more challenging. W. Grathwohl et al. claimed that their newly proposed sampler, Gibbs with Gradients, outperforms the current best samplers. We evaluated their claims on a series of randomly generated models as well as Ising and Potts models. The Gibbs with Gradients sampler is compared against the Gibbs sampler, and we empirically show that while Gibbs with Gradients decreases the autocorrelation of draws, the additional computational cost causes it to have a lower effective sample size per second, making it worse in practice.

Contents

1 Introduction and Motivation

The goal of this thesis is to review and evaluate the practical relevance of a new MCMC algorithm 'Gibbs with Gradients' [17].

Frequentist statisticians assume that the world is described by parameters that are fixed and unknown. These parameters get estimated by taking a random sample from the population and then computing some statistic over that sample. The computed value then becomes the estimate for the parameter. Usually the parameter is computed with a form of maximum likelihood estimation.

This approach has been successfully applied by statisticians and data scientists for centuries. Yet this approach does not allow us to reason probabilistically about the parameters. Bayesian statisticians take a different approach. They consider parameters not as fixed values, but as draws from a distribution. Since the parameters are drawn from distributions, we can then reason about them like random variables.

This approach has many advantages:

Confidence intervals. Bayesian statistics allows us to directly reason about confidence intervals. While Frequentist statistics can ask what the value is of some parameter, it cannot provide a confidence interval without performing multiple experiments. Bayesian statistics can directly give a confidence interval for the parameters as the parameters are assumed to follow some distribution.

Prior information. In general it is non-trivial to add prior information in Frequentists statistics. In Bayesian statistics we can simply assume that a parameter is drawn from some prior distribution. Adding prior information is then equal to multiplying it with the likelihood.

Hierarchical models. The values of the parameters are drawn from some distribution. This distribution might have more parameters itself, which are again drawn from some distribution. Bayesian statistics makes it very intuitive to model something very complex into simpler parts hierarchically.

Bayesian statistics is based on Bayes' theorem [24] and updating beliefs using this theorem is known as Bayesian inference. Probabilistic programming languages, like Stan [3], rely heavily on sampling to do Bayesian inference. To fully benefit from the advantages of Bayesian statistics we therefore need good samplers.

When, in Bayesian inference, we have an (unnormalized) posterior distribution, more often than not we would like to compute the expectation of some function f over that distribution. To compute $\mathbb{E}[f]$ we need the normalized posterior distribution. A naive approach to compute the normalization factor (of a discrete distribution) is to sum over each state and divide each probability by this sum. One of the biggest challenges we face when working with data is the curse of dimensionality. If we have n categorical variables with d options each, then our state space is of size n^d. Computing the normalization factor for a state space with size n^d becomes incredibly computationally expensive even for low values of n and d, and is in most cases intractable. A more feasible way to compute the posterior is to use the factorization of the distribution and then

propagate probabilities through this factorization, known as belief propagation [34]. However, this requires knowledge about the factorization of the distribution. This is not always available and would require a separate analysis of the distribution, if there even exists a useful factorization in the first place.

Instead of computing the posterior directly, we can use Monte Carlo integration to estimate the expectation $\mathbb{E}[f]$ over our distribution. Monte Carlo integration requires a sample from the posterior distribution. While sampling is a well studied problem in the continuous space [1, 8, 20, 30], samplers for discrete state spaces have received less attention. Nevertheless, if we want to estimate $\mathbb{E}[f]$ effectively using a sample, we need samplers that can effectively approximate the posterior distribution. In this thesis we will compare discrete samplers using the effective sample size, sampling speed, and scalability to high dimensions. Using these metrics we will try to answer the following question:

- How does Gibbs with Gradients compare to other discrete samplers, based on effective sample size, speed, and scalability to higher dimensions?

This naturally leads to the following sub-questions:

- How many draws can Gibbs with Gradients generate per second? How does this compare to other samplers?
- How many effective samples can be drawn per second? Is Gibbs with Gradients able to draw more effective samples in the same amount of time?
- How does an increasing number of dimensions affect the sampling speed and effective sample size?

1.1 Bayesian Inference and MCMC Samplers

Suppose we have some data $\boldsymbol{D}$ and we want to interpret $\boldsymbol{D}$ using a model with parameters $\boldsymbol{\theta}$. One way of doing this is to use the likelihood function $P(\boldsymbol{D} \mid \boldsymbol{\theta})$. We can find the parameters $\boldsymbol{\theta}$ such that this probability is maximized. This is known as maximum likelihood estimation.

Even though this approach is commonly used, it is somewhat counterintuitive. We care about the parameters yet we compute the probability of the data. In Bayesian statistics we take a different approach. Instead of computing $P(\boldsymbol{D} \mid \boldsymbol{\theta})$ we compute

$$P(\boldsymbol{\theta} \mid \boldsymbol{D}) = \frac{P(\boldsymbol{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{D})} \tag{1}$$

This equation consists of four parts: $P(\boldsymbol{\theta} \mid \boldsymbol{D})$ is the posterior; $P(\boldsymbol{D} \mid \boldsymbol{\theta})$ the likelihood; $P(\boldsymbol{\theta})$ the prior; and $P(\boldsymbol{D}) = \int_{\boldsymbol{\theta}} P(\boldsymbol{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta})$ the marginal likelihood.

The likelihood and the prior define an (unnormalized) probability mass function $f(\cdot)$. We can compute the estimate of any function $h : \mathcal{X} \to \mathcal{R}$ on discrete

state space $\mathcal{X}$ with respect to the posterior [41]:

$$\mathbb{E}_f[h(X)] = \frac{\sum_{\mathcal{X}} h(x)f(x)}{\sum_{\mathcal{X}} f(x)} \tag{2}$$

Unfortunately, computing this expected value is in general intractable. So, instead we turn to sampling to estimate the expected value of functions. More precisely we can use Monte Carlo integration to compute this estimate. Given draws $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from $f(\cdot)$ we can estimate $\mathbb{E}[h(X)]$ as

$$\mathbb{E}_f[h(X)] \approx \frac{1}{n} \sum_{i=1}^{n} h(x_i) \tag{3}$$

In a perfect world we would be able to get i.i.d. draws for any distribution. Unfortunately, in practice we have complex distributions from which it is really hard to sample. We will denote the i-th d-dimensional draw as $\boldsymbol{x}^i = \{x_1^i, x_2^i, \ldots x_d^i\}$. Instead of trying to get i.i.d. draws from the posterior, we will settle for draws that are correlated. That is, we produce a Markov Chain where draw $\boldsymbol{x}^{i+1}$ is dependent on draw $\boldsymbol{x}^i$. Because the draws are correlated our estimation error will decrease slower than for uncorrelated samples (see section 2.3.1).

We need to make sure that the probability of finding some state in our Markov Chain is equal to the probability in the posterior distribution $f(\cdot)$. To accomplish this we need the concept of a stationary distribution of a Markov Chain. Suppose we have a set of states with a transition probability matrix T, containing a transition probability for each pair of states. The stationary distribution of a Markov Chain is a probability vector s over the states of the chain such that $sT = s$. If we can get Markov Chain with stationary distribution $f(\cdot)$ and our samples converge to the stationary distribution, then we can take (correlated) draws from our posterior distribution.

There are two parts to this. Firstly, we need to design our transition probabilities in such a way that we get the right stationary distribution. A Markov chain has the right stationary distribution if it satisfies reversibility (also known as the detailed balance condition) [41]:

$$p(\boldsymbol{x})T(\boldsymbol{y} \mid \boldsymbol{x}) = p(\boldsymbol{y})T(\boldsymbol{x} \mid \boldsymbol{y}) \tag{4}$$

If we sum over all states x we get:

$$\sum_{\boldsymbol{x}} p(\boldsymbol{x})T(\boldsymbol{y} \mid \boldsymbol{x}) = p(\boldsymbol{y}) \sum_{\boldsymbol{x}} T(\boldsymbol{x} \mid \boldsymbol{y}) = p(\boldsymbol{y}), \tag{5}$$

which is equivalent to our definition of the stationary distribution.

Secondly, to converge, the Markov chain needs to be irreducible and aperiodic. It is periodic if, starting from a state s, there exists a t such that the probability of returning to state t is non-zero only at steps $n = t, 2t, \ldots$, otherwise it is aperiodic. It is irreducible if each state can be reached from any other state. If both these conditions hold then any Markov chain with a finite number of states is uniformly ergodic [41].

Definition. *[41] A Markov chain having stationary distribution $f(\cdot)$ is uniformly ergodic for some $\rho < 1$ and $M < \infty$ if*

$$||P^n(x, \cdot) - f(\cdot)|| < M\rho^n, \qquad n = 1, 2, 3, \ldots$$

While this gives us a guarantee that such a Markov chain will converge, both M and ρ might be values such that convergence can take a long time. In fact, the Central Limit Theorem tells us that the distribution of the sample mean tends toward the normal distribution as the sample size increases, regardless of the distribution from which we are sampling. Using the Central Limit Theorem, we know that the Markov Chain will converge with an error approximately equal to $1/\sqrt{n}$, when we have n i.i.d. draws. Since the draws are auto-correlated, the error decreases by $1/\sqrt{n_{\text{eff}}}$ as the effective sample size n_{eff} increases (see section 2.3.1). Slow convergence can have several causes some of which are: highly correlated draws (an issue of Gibbs sampling [33]), expensive draws, and lastly it might take a while before we reach the stationary distribution. The latter is known as the warm-up and it is common practice to discard some of the first draws.

The family of samplers that define transition probabilities and generate Markov Chains are known as Markov Chain Monte Carlo (MCMC) samplers. In this thesis we will compare two of such samplers. Namely, the Gibbs sampler and the Gibbs with Gradients sampler. We compare their performance using effective sample size, time per draw, and time per effective sample, which will be defined in section 2,

1.2 Structure of the Thesis

In this thesis we will compare the Gibbs with Gradients sampler with the Gibbs sampler. Section 2 explains the necessary background information on both the samplers and the metrics used to compare the samplers. Besides that, it provides examples of commonly used models with discrete parameters and some of their applications. In section 3 we propose a way to generate random models. These randomly generated models allow us to experiment on far more models. Section 4 first shows that, with high probability, our implementation of the samplers is corrent and then presents the results from the comparison between Gibbs and Gibbs with Gradients. Using these results we answer our research questions in the conclusions in section 5.

2 Literature Summary

2.1 Sampling and Markov Chain Monte Carlo

2.1.1 Metropolis-Hastings

One of the most commonly used MCMC algorithm is the Metropolis-Hastings algorithm [29, 19]. Metropolis-Hastings is not necessarily always used as a

sampler for discrete spaces, but is used as a standard framework by a lot of other samplers. It samples directly from the posterior distribution, and it does so in two steps.

The first part is generating a candidate draw. We define $q(\boldsymbol{x}' \mid \boldsymbol{x})$ as the proposal distribution. Now, given our previous sample $\boldsymbol{x}$, we draw a new candidate $\boldsymbol{x}' \sim q(\boldsymbol{x}' \mid \boldsymbol{x})$. An example for such a proposal distribution might be a normal distribution around $\boldsymbol{x}$. Sampling from the proposal distribution alone is not enough, as it might move the draws toward areas with arbitrarily small probabilities.

Therefore, in the second step an acceptance ratio is added. The algorithm should be more likely to accept samples with relatively high probability in the posterior distribution than those with low probability. To achieve this we add an acceptance ratio

$$A = \min\left(1, \frac{f(\boldsymbol{x'})}{f(\boldsymbol{x})} \frac{q(\boldsymbol{x} \mid \boldsymbol{x'})}{q(\boldsymbol{x'} \mid \boldsymbol{x})}\right),$$

where $f(\boldsymbol{x})$ is the probability mass of $\boldsymbol{x}$. With probability A we accept the newly proposed state, otherwise the new state is the current state. The pseudo-code can be seen in algorithm 1.

Algorithm 1: Metropolis-Hastings

Data: x the current state, $q(x' \mid x)$ the proposal distribution
Result: x' a new sample
$x' \sim q(x' \mid x)$
Accept with probability:

$$\min\left(1, \frac{f(\boldsymbol{x'})}{f(\boldsymbol{x})} \frac{q(\boldsymbol{x} \mid \boldsymbol{x'})}{q(\boldsymbol{x'} \mid \boldsymbol{x})}\right)$$

The detailed balance condition can easily be shown for the Metropolis-Hastings algorithm [41]. It is also irreducible under the mild assumption that

$$q(x' \mid x) > 0 \quad \forall (x, x') \in \mathcal{X} \times \mathcal{X} \tag{6}$$

and aperiodic if the current state is accepted with a non-zero probability [40].

2.1.2 Gibbs Sampling

The Gibbs sampler [12] is a special case of the Metropolis-Hastings algorithm where the proposed new state is accepted with probability 1. We will denote $\boldsymbol{\theta}_{\setminus i}$ as all parameters except for i. A Gibbs sampler is especially useful when it is easy to sample from $P(\boldsymbol{\theta}_i \mid \boldsymbol{\theta}_{\setminus i})$, but hard to sample from $P(\boldsymbol{\theta})$. Random Scan Gibbs Sampling works by first drawing a dimension d from a uniform proposal distribution $q(i)$ and then drawing a new value for x_d given all other dimensions. Gibbs Sampling is both irreducible and aperiodic, as a result it

converges. However, since it performs a random walk through the state space, convergence can be slow [33].

The pseudo-code for a (Random Scan) Gibbs Sampler is given in algorithm 2.

Algorithm 2: Random Scan Gibbs Sampling

Data: x the current state
Result: x' a new sample
$d \sim q(i)$
$x' \sim P(x_d \mid x_1, \ldots, x_{d-1}, x_{d+1}, \ldots, x_n)$

This algorithm can be tweaked in a couple of ways. First of all, we could say that changing all dimensions (sequentially) gives a new draw. Secondly, we could draw from more than one dimension at a time, known as Block-Gibbs. Finally, we could draw the dimension from any other proposal distribution, e.g. a uniform proposal distribution, known as Random Scan Gibbs. Combining these changes leads to many Gibbs-like algorithms, one of which will be discussed below.

Adaptive Gibbs
Adaptive Gibbs samplers [26] try to learn a proposal distribution $q(i)$ from which the dimension can be drawn. This is useful, as some dimensions might not have a lot of effect on the function we are trying to estimate. The updated algorithm is shown in algorithm 3. Adaptive Gibbs does require some more

Algorithm 3: Adaptive Gibbs with some update function R_n

Data: $X^{(n-1)} \ldots X^{(1)}$ the previous samples, $q(i)$ the current proposal distribution
Result: x' a new sample
$q'(i) \leftarrow R_i(q(i), X^{(n-1)}, \ldots, X^{(1)})$
$d \sim q'(i)$
$\boldsymbol{X}_d \sim P(x_d \mid x_1, \ldots, x_{d-1}, x_{d+1}, \ldots, x_n)$

thought as we both need to design an update rule R_n. This update rule takes the current proposal and a the previous draws and creates a new proposal distribution. Finding a good update rule is hard, but even more importantly a simple Adaptive Gibbs sampler might fail to converge [15]. An example of such an Adaptive Gibbs sampler is the Adaptive Random Scan Gibbs Sampler [5].

2.1.3 Gibbs with Gradients

A more recent approach to sampling from discrete distributions is Gibbs with Gradients [17]. It combines most of the ideas discussed in the samplers above

and adds gradient information to the proposal distribution. This is possible because most discrete distributions have an underlying continuous and differentiable distribution (see section 2.2), which is only evaluated on a discrete subset of values. One idea would be to sample from these underlying distributions as done in preceding work [31, 18]. We can transform draws from the continuous distribution to draws from the discrete distribution, but this has the problem that the continuous distribution may be arbitrarily difficult to sample from. Instead, Gibbs with Gradients directly uses the gradient information.

One of the problems that using gradient information tries to solve is that some dimensions might hardly ever change. Take for example the MNIST dataset, consisting of 255x255 images of handwritten digits. By far the largest part of the image is the (black) background. If our sampler proposes to change one of the dimensions which is nearly always black it will most likely be wasted computation, as the proposed value will be accepted with a very small probability.

Let us reconsider the proposal distribution $q(x \mid x')$. Note that while we have previously seen $q(i)$, which was a proposal distribution over the dimensions, $q(x \mid x')$ is a proposal distribution for a new state. In other words, what is the probability that we propose a new state x, given that we are currently in state x'. If we sample a single dimension we can rewrite it as $q(x \mid x') = \sum_i q(x \mid x', i)q(i)$ where $q(i)$ is a distribution over the dimensions. Here we split sampling a new candidate into: sampling the dimension $i \sim q(i)$; and proposing a change in dimension i. Sampling a dimension which hardly ever changes, like the background in the MNIST dataset, is wasteful. To tackle this problem Gibbs with Gradients uses an input dependent proposal distribution $q(i \mid x)$ to choose the dimensions to sample from. The algorithm uses gradient information about the distribution to create this more informed proposal distribution.

This gradient information is used in a locally-informed proposal. That is

$$q(x' \mid x) \propto e^{\frac{1}{\tau}(f(x')-f(x))}\mathbf{1}(x' \in H_m(x)), \tag{7}$$

where $H_m(x)$ is the Hamming ball of some size m around x, hence the 'locally-informed', $f(x)$ is the log-probability of state x. A Hamming ball $H_m(x)$ with maximum distance m is defined as the neighborhood of x:

$$H_m(x) = \{x' : d(x, x') \leq m, x' \in \boldsymbol{x}\}, \tag{8}$$

where $d(x, x')$ is the Hamming distance between the states x and x'. $H_m(x)$ is also known as a Hamming window of size m and τ is a temperature parameter which controls how much effect the local likelihood difference has. It has been shown that $\tau = 2$ gives an optimal acceptance rate [45]. We can approximate $f(x') - f(x)$ with a first order Taylor-series. For binary data we estimate it as

$$\tilde{d}(x) = -(2x - 1) \odot \nabla_x f(x), \tag{9}$$

and similarly for categorical data

$$\tilde{d}(x)_{ij} = \nabla_x f(x)_{ij} - x_i^T \nabla_x f(x)_i \tag{10}$$

In practice $\nabla_x f(x)$ can be computed using automatic differentiation [2]. Combining this with $\tau = 2$ gives us the estimated likelihood

$$q^{\nabla}(x' \mid x) \propto e^{\frac{\tilde{d}(x)}{2}} \mathbf{1}(x' \in H(x)), \tag{11}$$

which can be used in the standard Metropolis-Hastings algorithm.

For binary data and a Hamming window of size 1 this simplifies to choosing a dimension and then flipping that dimension. For categorical data we also need to choose a value. This is accomplished by making the proposal distribution a $D(K-1)$-way Softmax, where K is the number of options for the categorical parameters. The new value is then assigned using the deterministic $\text{flipdim}(x, i)$ as can be seen in algorithm 4. Choosing one out of the D dimension is equivalent as sampling from a categorical distribution over D choices:

$$q(i \mid x) = \text{Categorical}\left(\text{Softmax}\left(\frac{\tilde{d}(x)}{2}\right)\right) \tag{12}$$

The pseudo-code for Gibbs with Gradients is given in algorithm 4.

Algorithm 4: Gibbs with Gradients

Data: Unnormalized log-prob $f(\cdot)$, current sample x
Result: x' a new sample
Compute $\tilde{d}(x)$ (Eq. 9 if binary, Eq. 10 if categorical)
$q(i \mid x) \leftarrow \text{Categorical}\left(\text{Softmax}\left(\frac{\tilde{d}(x)}{2}\right)\right)$
$i \sim q(i \mid x)$
$x' \leftarrow \text{flipdim}(x, i)$
$q(i \mid x') \leftarrow \text{Categorical}\left(\text{Softmax}\left(\frac{\tilde{d}(x')}{2}\right)\right)$
Accept with probability:

$$\min\left(\exp(f(x') - f(x))\frac{q(i \mid x')}{q(i \mid x)}, 1\right)$$

2.2 Models and Applications

In order to evaluate samplers we need to have some distributions to sample from. In this section we will look at commonly used models and their distributions. Each of the models will have a distribution similar or equal to

$$p(\boldsymbol{x}) = \frac{1}{Z} e^{f(\boldsymbol{x})}, \tag{13}$$

where Z is a normalization constant such that the log-probability is $f(\boldsymbol{x}) - \log Z$.

The models have been selected based on their popularity. We would prefer the samplers to perform well on widely applicable models. Many of the selected

models have applications in biology, physics, natural language processing, and many other fields. Specific applications will be mentioned in the following subsections.

2.2.1 Ising Model

The Ising model [22] is originally designed as a mathematical model of ferromagnetism in statistical mechanics. The Ising model also finds use in modelling the motion of atoms [7], and neuroscience [42]. The model consists of discrete variables that represent magnetic dipole moments of atomic spins which can be in two states $+1$ or -1. Neighboring spins that agree have lower energy and the system tends to the lowest possible energy. Heat can disturb this tendency.

The 1-dimensional Ising model was solved by Ernst Ising in his thesis. It took almost twenty more years before a closed solution for the 2-dimensional Ising model was found by Lars Onsager [32]. Non-planar higher dimensional Ising models are known to be NP-complete [23]. Sampling the states can help us approximate the solution for higher dimensional Ising models, if we can do it efficiently.

The probability of being in a state x with energy $E(x)$, and constant β (the inverse temperature), is

$$f(x) = \frac{1}{Z} e^{-\beta E(x)}$$

The Ising model originally modelled the spins of atoms. Each atom has a spin $\sigma_i \in \{+1, -1\}$ and the total energy, with "interaction constant" J is

$$E(x) = -J \sum_{i,j} \sigma_i \sigma_j$$

2.2.2 Energy vs Probability

The Ising model defines an energy function E. This is because the Ising model, like many other models, is a so called energy-based model. Energy-based models (EBMs) are probabilistic models where the probability of being in a certain state is described by an energy function. As EBMs originate from physics, the energy in a system with state x is given by a Hamiltonian function H. Besides H, there is commonly some constant factor β, like temperature in the Ising model, which affects the probabilities. Using this Hamiltonian we can define the probability of being in any state as

$$f(x) = \frac{1}{Z} e^{-\beta H(x)} \tag{14}$$

2.2.3 Potts Model

The Ising model has a very strict assumption on the direction of the spins, namely that the atomic spin is either up or down. A generalization of the Ising model is the Potts model [37]. Instead of two states for each spin it consists of

q states. With $q = 2$ we get the Ising model and in the limit $q \to \infty$ it is known as the XY Model.

$$\sigma_n \in \{1, 2, \ldots, q\}$$

The energy in a state x then becomes

$$E(x) = -J \sum_{i,j} \delta(\sigma_i, \sigma_j)$$

where $\delta(\sigma_i, \sigma_j)$ is the Kronecker delta, that is

$$\delta(\sigma_i, \sigma_j) = \begin{cases} 1, & \text{if } \sigma_i = \sigma_j \\ 0, & \text{otherwise} \end{cases}$$

Cell Sorting

One of the applications of the Potts model is biological cell sorting. Cell sorting has been experimentally observed in vitro in embryonic cells. The sorting of cells does not involve any cell division, but only spatial rearrangement. Such rearrangement can be modelled using an extension of the Potts model [16].

One of the key differences with the normal Potts model is that we would like to keep the number of cells roughly constant. That is, for every state s and constant $V(s)$ we would like to have $\sum_i \delta(\sigma_i, s) - V(s) \approx 0$. We introduce a volume term $v(s) = \sum_i \delta(\sigma_i, s)$ which measures the number of cells which are in state s. The new energy function for a state x is now

$$E(x) = -J \sum_{<i,j>} \delta(\sigma_i, \sigma_j) - \sum_{s=1\ldots q} (v(s) - V(s))^2 \tag{15}$$

This leads to cell sorting through the rearrangement of cells rather than through cell growth. This model is more commonly known as the Cellular Potts Model.

2.2.4 Restricted Boltzmann Machine

A general Boltzmann Machine consists of two layers, one with hidden and one with visible nodes. The nodes have binary states and its energy function is therefore similar to that of an Ising model. Instead of J, a constant, we have a weight matrix w and we split our nodes into visible nodes $\boldsymbol{v}$ and hidden nodes $\boldsymbol{h}$. Each of the nodes has a bias $\boldsymbol{\beta}_i$. We only observe $\boldsymbol{v}$. The other variables w, $\boldsymbol{h}$ and $\boldsymbol{\beta}$ are unobserved. This gives the following energy function for a general Boltzmann Machine

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\left(\sum_{i<j} w_{ij} s_i s_j + \sum_i \boldsymbol{\beta}_i s_i \right), \tag{16}$$

where $\boldsymbol{s}$ is the union of the visible and hidden nodes. General Boltzmann Machines allows both visible and hidden nodes to be interconnected. In practice,

this makes it very hard to approximate the real distribution from which we observe data.

Instead, we will restrict the network from having any inter-layer connections. This is known as a Restricted Boltzmann Machine. For any state of $\boldsymbol{v}$ with corresponding bias $\boldsymbol{a}$; and $\boldsymbol{h}$ with corresponding bias $\boldsymbol{b}$ respectively, we can now compute its energy function as:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j, \tag{17}$$

giving us the following distribution:

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \tag{18}$$

Since the underlying structure of an RBM is bipartite, the hidden nodes are mutually independent given the visible nodes and vice versa. That is, the conditional independencies are

$$\begin{aligned} P(\boldsymbol{v} \mid \boldsymbol{h}) &= \prod_i P(v_i \mid h), \\ P(\boldsymbol{h} \mid \boldsymbol{v}) &= \prod_j P(h_j \mid v) \end{aligned} \tag{19}$$

The individual probabilities of a node are dependent on its own bias and the weight of the connections between nodes. We then get a valid probability with the sigmoid function. That is, the probabilities are given by

$$\begin{aligned} P(v_i \mid \boldsymbol{h}) &= \sigma\left(a_i + \sum_j w_{i,j} h_j\right) \\ P(h_j \mid \boldsymbol{v}) &= \sigma\left(b_j + \sum_i w_{i,j} v_i\right) \end{aligned} \tag{20}$$

This allows for easy training using contrastive divergence [4].

2.2.5 Conditional Random Fields

Labelling a sequence of observations is a task which is used in many fields. Some examples of applications are protein sequences [9], computational linguistics [27] and part-of-speech tagging [38].

In CRFs the probability of seeing a label sequence $\mathbf{y}$ given observations $\mathbf{x}$ can be defined as a product of potential functions [25]

$$\exp\left(\sum_j \lambda_j t_j(y_{i-1}, y_i, \boldsymbol{x}, i) + \sum_k \mu_k s_k(y_i, \boldsymbol{x}, i)\right), \tag{21}$$

where t_j are transition feature functions and s_k are state feature functions. Allowing the potential functions to take the entire observation sequence as an input allows for arbitrary relations between the input.

To define feature functions, we use a set of real-valued features $b(\boldsymbol{x}, i)$ modelling some characteristic of the data. An example $b(\boldsymbol{x}, i)$ could be:

$$b(\boldsymbol{x}, i) = \begin{cases} 1, & \text{if } x_i \text{ is uppercase} \\ 0, & \text{otherwise} \end{cases}$$

We could then define a feature function:

$$s_k(y_i, \boldsymbol{x}, i) = \begin{cases} b(\boldsymbol{x}, i), & \text{if } y_i \text{ is a proper noun} \\ 0, & \text{otherwise} \end{cases}$$

The only difference between t_j and s_k is that s_k does not consider y_{i-1}. As a result, it only gives information about the probability of seeing y_i given the observed sequence $\boldsymbol{x}$.

We can simplify the notation by saying that $s_k(y_i, \boldsymbol{x}, i) = s_k(y_{i-1}, y_i, \boldsymbol{x}, i)$ and define $f_j(y_{i-1}, y_i, \boldsymbol{x}, i)$ as being either a transition or a state function. The probability can then be written as

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\lambda}) = \frac{1}{Z} e^{\sum_j \lambda_j F_j(\boldsymbol{y}, \boldsymbol{x})}, \tag{22}$$

where $F_j(\boldsymbol{y}, \boldsymbol{x}) = \sum_j f_j(y_{i-1}, y_i, \boldsymbol{x}, i)$ and Z is a normalization factor.

2.2.6 Mixture Models

Mixture models assume that the distribution consists of K mixture components. It splits $f(\boldsymbol{x})$ into $\sum_{x'} f(\boldsymbol{x}, x')$, where x' is a categorical parameter with K possible values. Often these values are unobserved (latent) variables. Marginalizing out a parameter is convenient when we suspect that different values of x' follow a different distribution. Because of this, mixture models are mostly used to perform clustering, a form of unsupervised learning.

The final distribution is a mixture of the K components, such that

$$f(\boldsymbol{x}) = \frac{1}{Z} \sum_{x'} f(x') f(\boldsymbol{x} \mid x') \tag{23}$$

Mixture models find many applications, some examples are text clustering [44], speaker identification [39], graph structure discovery [6], gene subset selection [28] and time series clustering [11].

While all of the mentioned models are widely used, each with its own applications, in our experiments we will focus on Ising and Potts models as well as randomly generated models, which we will discuss in section 3. Ising and Potts models are known to be hard to sample from. Performing well on these models would therefore indicate that the sampler will, most likely, also performs well on other models.

2.3 Evaluation Metrics

In order to evaluate Gibbs with Gradients against other discrete samplers we will need to compare some metrics of performance. We would like to have a sampler which efficiently explores the posterior distribution. MCMC samplers take correlated draws by taking local steps in the distribution. These correlated draws can have less information than i.i.d. draws. In the most extreme case where we take n samples $\{x_1, ..., x_i\}$ where $x_i = x_j$ for all pairs of i and j we really only have one effective sample. In general if our n samples are correlated, we need to adjust for that correlation. This adjusted sample size is known as the effective sample size (ESS). Since a perfect sampler has an ESS equal to, or even exceeding, the sample size, this will be our first evaluation metric. In addition, a high ESS is only useful if the sampler is able to generate draws at an adequate speed, and lastly we would want our samplers to scale well to high-dimensional models.

2.3.1 Effective Sample Size

If we have independent draws we can bound the uncertainty in estimates by the central limit theorem. However, if our samples are correlated we need to use the number of effective samples (N_{eff}) instead. Using the central limit theorem we can now see that the estimation error decreases with $1/\sqrt{N_{\text{eff}}}$ rather than $1/\sqrt{N}$. The effective sample size is directly affected by the autocorrelation.

The autocorrelation is a measure of how correlated a sequence is with respect to itself at some lag t. We denote this ρ_t. The effective sample size of N samples generated by a process with autocorrelations ρ_t is defined by

$$N_{\text{eff}} = \frac{N}{\sum_{t=-\infty}^{\infty} \rho_t} = \frac{N}{1 + 2\sum_{t=1}^{\infty} \rho_t} \tag{24}$$

In practice, it is not tractable to compute ρ_t as it requires integrating the probability function. Instead, the autocorrelation has to be estimated. Fortunately, we can estimate each time lag at the same time using a fast Fourier transform [13].

2.3.2 Autocorrelation Time

While high ESS is mostly preferred over a low ESS, speed has a significant impact in practice. We need to consider both how many draws do we need to get a high enough ESS and how fast we can generate a single draw.

An alternative to the effective sample size, which we chose not to use, is the autocorrelation time [10]. It is a direct measure of the number of evaluations needed before a sampler produces independent draws. A low autocorrelation time means that we need fewer draws to produce a representative sample. More formally, the autocovariance of a sequence $X(t)$ is

$$C_f(T) = \lim_{t \to \infty} \text{cov}[f(X(t+T)), f(X(t))], \tag{25}$$

where $C_f(T) \to 0$ measures the number of draws that must be taken to ensure independence, assuming that $C_f(T)$ converges to 0 for some number of draws. The autocorrelation time is defined as [10]:

$$\tau_f = \sum_{T=-\infty}^{\infty} \frac{C_f(T)}{C_f(0)} = 1 + 2\sum_{T=1}^{\infty} \frac{C_f(T)}{C_f(0)} \tag{26}$$

Where $C_f(T)$ for a Markov chain can be estimated by[1]

$$C_f(T) \approx \frac{1}{M-T} \sum_{m=1}^{M-T} [f(X(T+m)) - \langle f \rangle][f(X(m)) - \langle f \rangle], \tag{27}$$

where $\langle f \rangle$ is the mean of $f(X(m))$ for all m in the sequence.

2.3.3 Time per Draw

The effective sample size only measures how well we can explore the distribution. It does not tell us anything about the speed at which it is explored. A combination of a high effective sample size with a slow sampler might be worse than a sampler with a slightly worse effective sample size, but with faster draws. As the error decreases by $1/\sqrt{N_{\text{eff}}}$ and the speed at which we can take an effective sample is $t\frac{N}{N_{\text{eff}}}$, we need both a low t and a high effective sample size to efficiently reduce the error. We will measure the time per draw in milliseconds. From this time, in combination with the effective sample size, we compute the time per effective sample.

2.3.4 Dimensions

One of the reasons we need sampling in the first place is because of high dimensional models. It is crucial that samplers can deal with this. For example, modelling protein sequences might include hundreds of amino acids [21], quickly leading to the inability to efficiently sample from their distributions. Being able to sample from such models allows us to train and examine larger models, possibly leading to large improvements.

Therefore, each of the evaluation metrics mentioned above will be computed on an increasing number of dimensions, until it becomes infeasible to take enough draws. This allows us to compare the scalability of different samplers. Which, with the ever growing size of models, is an incredibly important aspect.

3 Random Models

In addition to the models discussed in section 2.2, we propose to generate random models. Random models can be used to study the behavior of sampling

[1] Implemented in a python package https://github.com/dfm/acor

algorithms on a large number of different models. This will allow us to subjectively assess how well a sampling algorithm performs. For a single algorithm it is preferable to perform well on a large set of models rather than a specific one.

The first thing to notice is that a model is simply a distribution over the parameters $\boldsymbol{\theta}$ and the data. We will assume that there is no observed data. Therefore our random models will be distributions over the parameters only. Such a distribution can be factorized into k factors, where factor ψ_i is dependent on $\boldsymbol{\theta}_i \subseteq \boldsymbol{\theta}$:

$$f(\boldsymbol{\theta}) = \frac{1}{Z} \cdot \psi_1(\boldsymbol{\theta}_1) \cdot \psi_2(\boldsymbol{\theta}_2) \cdot \ldots \cdot \psi_k(\boldsymbol{\theta}_k)$$

We generate $f(\boldsymbol{\theta})$ randomly in two parts. We first generate a random factorization over the parameters. Secondly, we generate random functions $\psi_i(\boldsymbol{\theta}_i)$ for all i. The normalization constant Z will remain unknown in the most cases, but can be computed using brute-force on smaller models. We will use this to test the correctness of the samplers.

3.1 Generating random factorizations

There are many ways to represent the conditional dependence structures between random variables, one of which is the use of probabilistic graphical models (PGMs). A PGM represents conditional (in)dependencies in a directed or undirected graph. Instead of generating a random factorization directly, we can instead generate such a graph. As there are many kinds of probabilistic graphical models, we will discuss some of the options below.

3.1.1 Bayesian Networks

Bayesian networks (or directed graphical models) are directed acyclic graphs which encode the factorization of the joint probability distribution. More formally, a bayesian network is a pair $\mathcal{B} = (G, \Gamma)$ such that

- G is a directed acyclic graph with nodes representing random variables $\boldsymbol{x}$
- $\psi = \{\psi_{x_i} \mid x_i \in \boldsymbol{x}\}$ is a set of assessment functions

This defines a joint probability distribution

$$p(\boldsymbol{x}) = \prod_{x_i \in \boldsymbol{x}} \psi_{x_i}(x_i \mid \boldsymbol{\rho}(x_i))$$

with $\boldsymbol{\rho}(x_i)$ the set of parents of x_i in G. A downside of Bayesian networks is that reading of conditional independencies is not trivial, as it requires the concept of d-separation [35].

3.1.2 Markov Random Fields

Markov random fields (or undirected graphical models) are undirected graphs which, again, encode the factorization of the joint probability distribution. In a

Markov random field, a set of variables A is conditionally independent of another set B given a set of variables S, if all paths from A to B pass through S. The conditional independencies are encoded by definition. Given all maximal cliques C in the Markov random fields gives its factorization:

$$p(\boldsymbol{x}) = \frac{1}{Z} \cdot \prod_{\boldsymbol{c} \in C} \psi_{\boldsymbol{c}}(\boldsymbol{c})$$

However, finding all maximal cliques is not trivial either. Instead, we want to find a representation in which we easily define the factors and read off independencies. To do so, we will look at a third representation.

3.1.3 Factor Graphs

A factor graph is a bipartite graph $G = (V, F, E)$ of parameters V, factors F and edges E, such that each edge connects a factor to a variable. It provides a factorization of a distribution $p(\boldsymbol{\theta})$ into factors over sets of parameters:

$$f(\boldsymbol{\theta}) = \prod_{f_i \in F} f_i(\boldsymbol{\theta}_{f_i})$$

where $\boldsymbol{\theta}_{f_i}$ is the set of parameters which are connected to f_i by an edge in E. This allows a factor graph to give us direct control over both the independencies and the factors. Since we need to know the independencies to generate factors, we care mostly about the factorization of the probability distribution. For this purpose factor graphs suit our needs perfectly.

We can randomly generate a factor graph using the Erdős-Rényi-Gilbert model [14]. That is, we take n parameters and m factors. Then, we add an edge between each of the pairs of parameters and factors with probability p. Two examples of generated factor graphs can be seen in figure 1.

3.2 Generating random factors

The second, and arguably harder, part is to generate random functions. These functions have to be differentiable as Gibbs with Gradients uses gradient information. Besides that, it would be infeasible to have completely random functions. Instead, the functions will be generated in a form such that they are an element of the exponential family.

In statistics the exponential family [36] is a parametric set of probability distributions. That is, a set of probability distribution where the number of parameters is finite. Exponential families provide a general framework for selecting a parameterization of a parametric family of distributions, in terms of natural parameters. Different choices for the natural parameters result in different distributions.

The exponential family of distributions with parameters $\boldsymbol{\theta}$ and variables $\boldsymbol{x}$ is a set of probability distributions which can be expressed in the form

$$f(\boldsymbol{x} \mid \boldsymbol{\theta}) = h(\boldsymbol{x}) \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(\boldsymbol{x}) - A(\boldsymbol{\eta})\right) \tag{28}$$

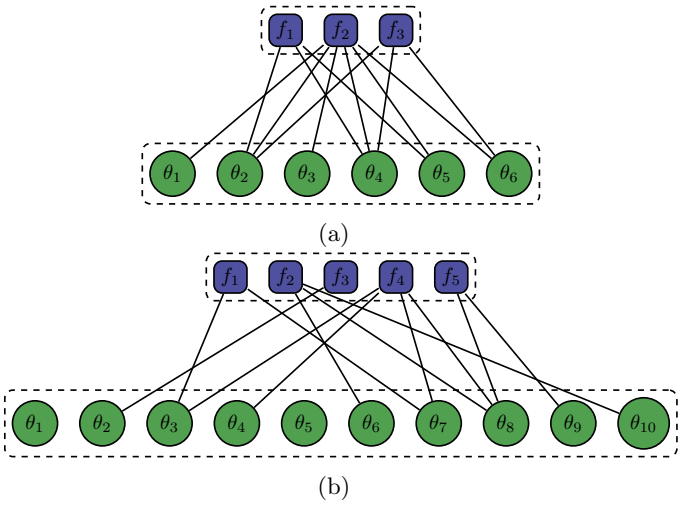


Figure 1: (a) A factor graph generated with $n = 6$, $m = 3$ and $p = 0.75$. (b) A factor graph generated with $n = 10$, $m = 5$ and $p = 0.5$

where $\boldsymbol{\eta}(\boldsymbol{\theta})$ are known as the natural parameters. In this form $A(\boldsymbol{\eta})$ is a log-normalization factor (more commonly known as the log-partition function), which can easily be shown as follows:

$$\begin{aligned} f(\boldsymbol{x} \mid \boldsymbol{\theta}) &= h(\boldsymbol{x}) \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(\boldsymbol{x}) - A(\boldsymbol{\eta})\right) \\ &= h(\boldsymbol{x}) \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(\boldsymbol{x})\right) \exp\left(-A(\boldsymbol{\eta})\right) \\ &= \frac{h(\boldsymbol{x}) \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(\boldsymbol{x})\right)}{\exp\left(A(\boldsymbol{\eta})\right)} \\ &= \frac{h(\boldsymbol{x}) \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(\boldsymbol{x})\right)}{\int h(\boldsymbol{x}) \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(\boldsymbol{x})\right) \nu(d\boldsymbol{x})} \end{aligned} \tag{29}$$

therefore

$$f(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z} \cdot h(\boldsymbol{x}) \exp\left(\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(\boldsymbol{x})\right)) \tag{30}$$

where $\log Z = A(\boldsymbol{\eta})$.

Many of the distributions commonly found in data science can be rewritten as an exponential family distribution. This provides us with a useful generalization of those distributions. Some examples can be seen in table 1.

Our goal is to generate a random function of this form. We assume that the data is unknown, this effectively makes $h(\boldsymbol{x})$ a constant and $T(\boldsymbol{x})$ a vector of constants. We can therefore focus on generating $\boldsymbol{\eta}(\boldsymbol{\theta})$, the natural parameters. Each likelihood will be a linear combination of the natural parameters.

We can clearly see that we would (at a minimum) need the following operators: addition and subtraction $f + g$, $f - g$; logarithm: $\log f$; negation: $-f$; inverse: f^{-1}; square: f^2; multiplication: fg; and constants: c.

Distribution	Parameters $\boldsymbol{\theta}$	Natural parameters $\boldsymbol{\eta}$
Bernoulli	p	$\log \frac{p}{1-p}$
Poisson	λ	$\log \lambda$
Exponential	λ	$-\lambda$
Normal	μ, σ^2	$\begin{bmatrix} \frac{\mu}{\sigma^2} \\ -\frac{1}{2\sigma^2} \end{bmatrix}$

Table 1: Some examples of popular distributions with their natural parameters.

If we define $\boldsymbol{\eta}(\boldsymbol{\theta})$ as

$$\boldsymbol{\eta}(\boldsymbol{\theta}) = \begin{bmatrix} \eta_1(\boldsymbol{\theta}) \\ \eta_2(\boldsymbol{\theta}) \\ \vdots \\ \eta_k(\boldsymbol{\theta}) \end{bmatrix} \tag{31}$$

then $\eta_i(\boldsymbol{\theta})$ should at least require $\boldsymbol{\theta}_i$ to be a part of the function. $\eta_i(\boldsymbol{\theta})$ is therefore a generated tree of operations containing $\boldsymbol{\theta}_i$ as a leaf. Repeating this for all k parameters results in $\boldsymbol{\eta}(\boldsymbol{\theta})$. To actually generate the functions $\eta_i(\boldsymbol{\theta})$ we will use a context-free grammar.

A context-free grammar G is a 4-tuple $G = (M, T, R, S)$. M is the set of non-terminal symbols, T is the set of terminal symbols (in our case the operations), R is the set of production rules, and lastly S is the start symbol. A context-free grammar has production rules in the form of

$$S \to \alpha$$

where S is a non-terminal (start) symbol and α consists of non-terminal and terminal symbols. A probabilistic context-free grammar adds a fifth symbol P. P is the set of probabilities associated with our production rules. The probabilities of the production rules of a non-terminal symbol always sum to one. A probabilistic context-free grammar allows us to generate a tree of operations.

The following PCFG will be used to generate our random functions:

$$
\begin{array}{lcl}
\eta_i & \rightarrow & \text{FunctionWithParam [1.0]} \\
\text{FunctionWithParam} & \rightarrow & \text{Add Function FunctionWithParam [0.08]} \\
 & | & \text{Add FunctionWithParam Function [0.08]} \\
 & | & \text{Subtract Function FunctionWithParam [0.08]} \\
 & | & \text{Subtract FunctionWithParam Function [0.08]} \\
 & | & \text{Multiply Function FunctionWithParam [0.08]} \\
 & | & \text{Multiply FunctionWithParam Function [0.08]} \\
 & | & \text{Log FunctionWithParam [0.08]} \\
 & | & \text{Square FunctionWithParam [0.08]} \\
 & | & \text{Inverse FunctionWithParam [0.08]} \\
 & | & \text{Negate FunctionWithParam [0.08]} \\
 & | & \text{Parameter [0.2]} \\
\text{Function} & \rightarrow & \text{Add Function Function [1/14]} \\
 & | & \text{Subtract Function Function [1/14]} \\
 & | & \text{Multiply Function Function [1/14]} \\
 & | & \text{Log Function [1/14]} \\
 & | & \text{Square Function [1/14]} \\
 & | & \text{Inverse Function [1/14]} \\
 & | & \text{Negate Function [1/14]} \\
 & | & \text{Parameter [4/14]} \\
 & | & \text{AnyParameter [2/14]} \\
 & | & \text{Constant [1/14]}
\end{array}
$$

where **Parameter** is $\boldsymbol{\theta}_i$ for η_i and **AnyParameter** is an uniformly chosen parameter. A constant is either -1, 1 or 2, each with probability 1/3. The probabilities of each rule are determined experimentally balancing freedom with depth of the trees.

We can now generate a wide variety of functions, for example the natural parameters of a normal distribution with $\theta_1 = \mu$ and $\theta_2 = \sigma$:

$\eta_1(\boldsymbol{\theta}) = \text{Multiply (Parameter) (Inverse (Square (AnyParameter)))}$

$\eta_2(\boldsymbol{\theta}) = \text{Negate (Inverse (Multiply (Constant) (Square (Parameter))))}$

or the natural parameter of the exponential distribution

$$\eta_1(\boldsymbol{\theta}) = \text{Negate (Parameter)}$$

which will be generated with probability

$$
\begin{aligned}
P(\eta_1) &= P(\text{FunctionWithParam}) \cdot P(\text{Negate FunctionWithParam}) \cdot P(\text{Parameter}) \\
&= 1.0 \cdot 0.08 \cdot 0.2 = 0.016
\end{aligned}
$$

$\boldsymbol{\eta}(\boldsymbol{\theta})$
$\begin{bmatrix} \boldsymbol{\theta}_0 - \boldsymbol{\theta}_0^{-1} \\ (\boldsymbol{\theta}_1^2 + \boldsymbol{\theta}_1)^2 \end{bmatrix}$
$\begin{bmatrix} \boldsymbol{\theta}_0 \\ 2\boldsymbol{\theta}_1 \end{bmatrix}$
$\begin{bmatrix} \boldsymbol{\theta}_0 * \boldsymbol{\theta}_0 \\ \boldsymbol{\theta}_0 \cdot \log\log(\boldsymbol{\theta}_1 + \boldsymbol{\theta}_1) \cdot \boldsymbol{\theta}_1 \end{bmatrix}$

Table 2: Randomly generated functions with two parameters using the PCFG defined above

More examples can be seen in table 2.

To summarize, for a set of parameters $\boldsymbol{\theta}$ we can generate a random factor by generating

- h: a random number
- $\boldsymbol{T}$: a vector of random number with length $|\boldsymbol{\theta}|$
- $\boldsymbol{\eta}(\boldsymbol{\theta})$: a function containing η_0 up to $\eta_{|\boldsymbol{\theta}|}$

The full procedure to generate a random factor can be seen in algorithm 5.

Algorithm 5: Generating a random factor

Input: Number of parameters $|\boldsymbol{\theta}|$
Result: $\boldsymbol{\eta}(\boldsymbol{\theta})$
$h \sim \mathcal{N}(0, 1)$
$\boldsymbol{T} \leftarrow$ random vector of numbers drawn from $\mathcal{N}(0, 1)$
for $i \in [0, |\boldsymbol{\theta}|]$ **do**
  $f \leftarrow$ generate function using PCFG
  $\eta_i \leftarrow f$
end
return $\lambda\boldsymbol{\theta} \rightarrow h \exp\left(\prod_i \eta_i(\boldsymbol{\theta}) \cdot \boldsymbol{T}\right)$

3.3 Generating random models

Now that we can both generate a random factorization and random factors, we can put it all together. First, we generate a random factor graph with n parameters and m factors, each of which are connected with chance p. Then, for each factor in the factor graph, we generate a random factor from the exponential family. Lastly we return a function which takes all parameters $\boldsymbol{\theta}$ and returns the product of the factors, where each factor is computed using $\boldsymbol{\theta}_i \subseteq \boldsymbol{\theta}$, which is the subset of parameters connected to the factor in the factor graph. The procedure of generating a random model can be seen in algorithm 6.

Algorithm 6: Generating a random model

Result: A random model with parameters $\boldsymbol{\theta}$
Input: n, m, p
graph $\leftarrow$ generate bipartite graph with n parameters and m factors, connected with probability p
for *factor node* $\in$ *graph* **do**
  factor $\leftarrow$ generate factor with $\boldsymbol{\theta}_i \subseteq \boldsymbol{\theta}$ (algorithm 5)
end
return $\lambda\boldsymbol{\theta} \rightarrow \prod_{f \in \text{factors}} f(\boldsymbol{\theta}_i)$

3.4 Limitations of the generated models

While generating models randomly allows us to increase the number of models to sample from, they have some limitations. Firstly, our PCFG allows for the generation of functions which are undefined for a subset of the parameter space. For example, $f(\boldsymbol{\theta}) = \log(\theta_0 - \theta_0)$ will always result in $\log(0)$ which is undefined. To simplify the process of generating models we will assume that $e^{-\infty} = 0$. Secondly, divisions by zero can occur. A simple example of this is $f(\boldsymbol{\theta}) = 1/\theta_0$. We will consider this an invalid model, which we discard.

4 Results

In this section we will discuss the results we gathered on the Gibbs and Gibbs with Gradients sampler. It consists of four subsections.

Section 4.1 discusses the correctness of the implementation of the samplers. If the samplers are correct they will approximate the posterior distribution. We test whether the samplers do indeed sample from the posterior distribution using Chi-Square tests.

In section 4.2 the performance of both samplers on Ising models are analyzed. We compute the number of effective samples and time per effective sample on Ising models with increasing dimensionality and different values of J. Similarly, section 4.3 discusses the same measures computed on Potts models with both an increasing number of dimensions and categories per parameter.

Lastly, section 4.4 discusses the performance of the samplers on randomly generated models. We compute the number of effective samples and time per effective sample on models with a varying amount of parameters and factors. If one sampler is truly better than the other we would expect to see these results in our randomly generated models.

Both the models and the samplers are implemented in python using pytorch[2]. We ran all experiments on a 6-core Ryzen 5600X processor running at a base clock of 3.7GHz using 32GB of RAM, using Windows 10 version 21H2 as the operating system.

[2] https://pytorch.org/

4.1 Correctness

Proving correctness of any implementation of a probabilistic algorithm is non-trivial. In this section we will statistically show that both the Gibbs and the Gibbs with Gradients sampler, truly sample from the posterior distribution. To do this we will perform a Chi-Square test on both an Ising model and some randomly generated models. If the samplers draw from the posterior we would expect the estimated probability mass to approach the posterior as the number of draws goes to infinity.

A Chi-Square (goodness-of-fit) test is a statistical test, where the null hypothesis assumes that there is no difference between the expected and the observed value. The alternative hypothesis assumes that there is a significant difference between the observed and the expected value. First we compute the χ^2 value, defined as

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i} \tag{32}$$

where O_i is the ith observed value and E_i is the ith expected value. In our case O_i is the number of times we have drawn the ith state, and E_i is the number of draws we expected for state i.

We compute the expected values using brute-force. That is, we compute the normalization constant Z by summing over all possible states. This limits our ability to test the algorithms on larger models, but will be sufficient to show, with high confidence, that we have a correct implementation.

4.1.1 Correctness of the Gibbs Sampler

As the (Random Scan) Gibbs Sampler is a relatively simple algorithm we will start with it. Let us first sample from a relatively small model. We sample from a 2 by 2 Ising model with $J = 0.1$. As this model only has 16 distinct states, we can easily visualize how the posterior gets estimated in figure 2. With only 100 draws the estimated probability mass does not resemble the true posterior very well. As we increase it to 1000 we can clearly see the improvement. At 10000 draws the estimated probability mass is nearly identical to the true posterior.

While sampling from this 2 by 2 Ising model seems to indicate that everything should work correctly, we will still perform a Chi-Square test on a larger 4 by 4 Ising model. With 2^{16} states it is very unlikely that we would get anywhere near the posterior with an incorrect implementation. We sample from a 4 by 4 Ising model with $J = 0.1$. We run 10 chains and take 10.000 draws per chain, giving us a 100.000 draws in total. Then the count of each state is averaged over all chains. This results in our observed values. Since our model has 2^{16} states, the Chi-Square test is performed with $2^{16} - 1$ degrees of freedom. The critical value for $\alpha = 0.01$ is 66380. Using the observed values as described we get $\chi^2 = 21345.5$. This value is well below the critical value and we can therefore accept the hypothesis that the Gibbs sampler samples from the posterior.

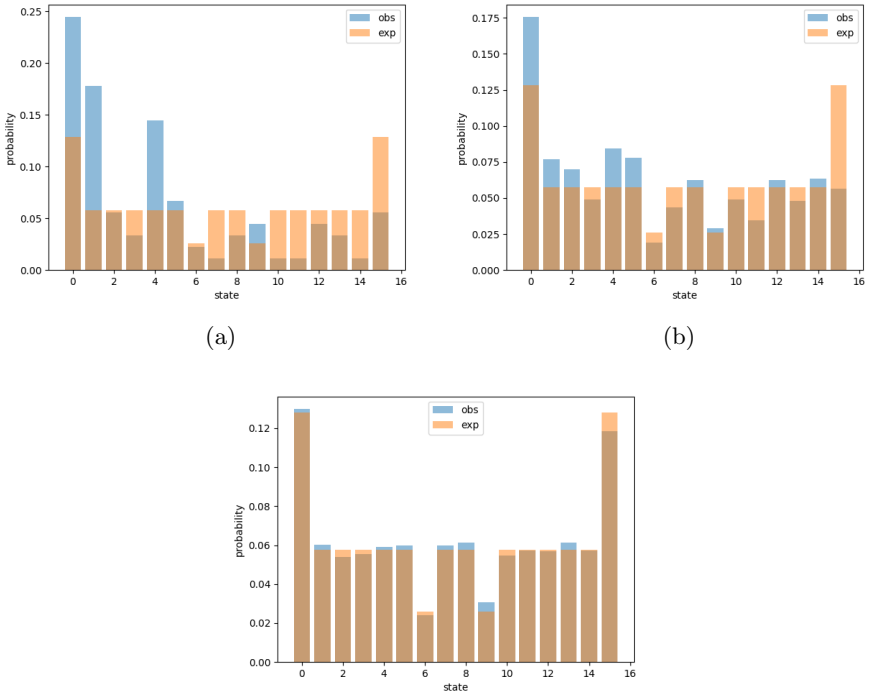
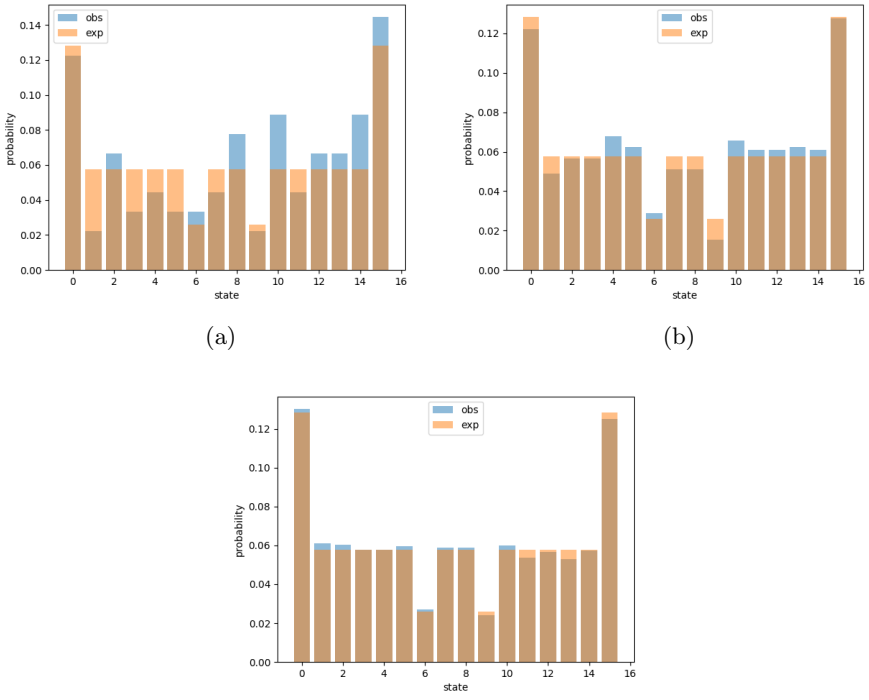


Figure 2: The true distribution of an Ising model with $J = 0.1$ vs. the estimated distribution using draws from the Gibbs sampler. (a) 100 draws, (b) 1000 draws, (c) 10.000 draws.

4.1.2 Correctness of the Gibbs with Gradients Sampler

Similar to the Gibbs Sampler we will first (informally) show that the Gibbs with Gradients sampler can approximate the posterior of a 2 by 2 Ising model. In figure 3 we can see that at 100 draws the observed values do not match the posterior very well. It does however provide a much better approximation than the Gibbs sampler at 100 draws. The approximation is a lot better at a 1000 draws and very close to the true posterior at 10.000 draws.

To prove the correctness of the Gibbs with Gradients sampler we will again use a Chi-Square test with $\alpha = 0.01$. The setup is equivalent to that of the Gibbs sampler. We sample using 10 chains with 10.000 draws per chain and average the observed counts. The critical value for $\alpha = 0.01$ and $2^{16} - 1$ degrees of freedom is 66380. For the Gibbs with Gradients sampler we get $\chi^2 = 7783.3$, which is again smaller than the critical value 66380. We therefore accept the hypothesis that the Gibbs with Gradients sampler samples from the true posterior distribution.



(a) (b)

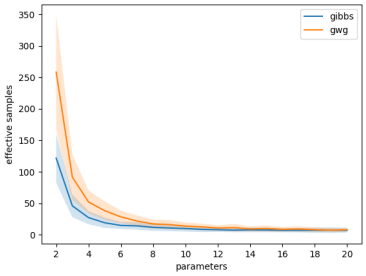
(c)

Figure 3: The true distribution of an Ising model with $J = 0.1$ vs. the estimated distribution using draws from the Gibbs with Gradients sampler. (a) 100 draws, (b) 1000 draws, (c) 10.000 draws.

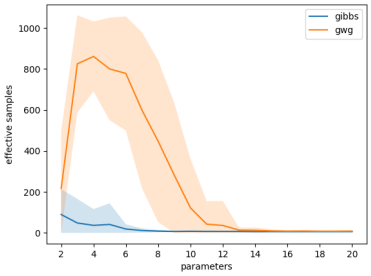
4.2 Sampling from Ising Models

An Ising model is a binary energy-based model arranged in a graph (usually a lattice). The correlation between neighbors in the graph depends on the interaction constant J. In our experiment we compute the effective sample size for various values of J. If J is low the neighboring nodes are (close to) independent and as J increases they become more dependent.

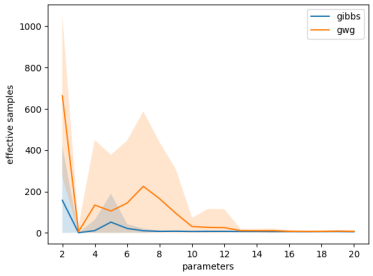
In our experiment we create a $N \times N$ lattice Ising model where N is between 1 and 20. We take $J = 0.1$ until $J = 0.9$ incrementing it by 0.1 each step. For both Gibbs and Gibbs with Gradients we take 1000 draws for each combination of N and J and compute the effective sample size and time per draw.



(a) J = 0.1



(b) J = 0.5



(c) J = 0.9

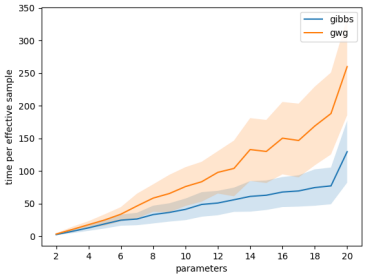
Figure 4: The number of effective samples (out of 1000 draws) from a Gibbs Sampler and a Gibbs with Gradients Sampler from 1 to 20 dimensions and $J \in \{\ 0.1, 0.5, 0.9\ \}$

Figure 4a shows the effective sample size at $J = 0.1$. At this value Gibbs with Gradients outperforms the Gibbs Sampler on low dimensions by a small amount. Both samplers perform very poorly, even with as little as 5 dimensions. At $J = 0.9$ Gibbs with Gradients starts outperforming the Gibbs sampler up until $N = 13$, as can be seen in figure 4c. The real difference between the two samplers can be seen at $J = 0.5$, when the nodes are only partially correlated. At this value Gibbs with Gradients easily outperforms the Gibbs Sampler in terms of the effective sample size (figure 4b). Both samplers clearly struggle with N greater than 10. That might, however, not be that surprising given that $N = 10$ gives 2^{100} states. The full result can be seen in appendix A.

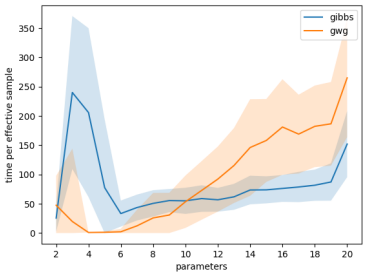
More interesting than the effective sample size, is the time per effective sample size. At $J = 0.5$ Gibbs with Gradients, which outperformed Gibbs in terms of the effective sample size, only has a lower time per effective sample up until 10 dimensions (figure 5b). After that Gibbs becomes the better sampler when it comes to time per effective sample. If we compare figure 4b to figure 5b this makes sense as both samplers struggle to sample from the distribution, but Gibbs with Gradients has the additional cost of computing the gradient information. In general, this makes the Gibbs with Gradients sampler significantly

slower on Ising models. Looking at $J = 0.1$ and $J = 0.9$ Gibbs with Gradients performs worse on time per effective sample regardless of the number of dimensions (figure 5a, 5c).

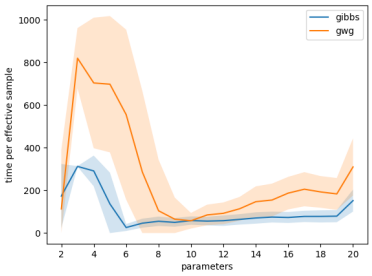
Gibbs with Gradients outperforms Gibbs when the atomic spins (or nodes) are partially correlated. That is, it can produce effective samples more efficiently than Gibbs around $J = 0.5$. When the atomic spins are highly correlated, or barely correlated at all, it quickly loses it advantage to the increased cost of computing the gradient information.



(a) J = 0.1



(b) J = 0.5



(c) J = 0.9

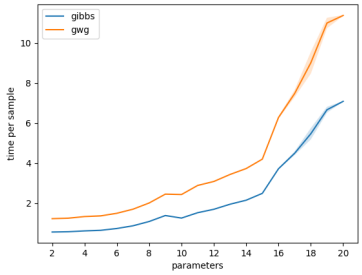
Figure 5: The time per effective sample for both the Gibbs and the Gibbs with Gradients sampler with a lattice size from 1 up to 20 and $J \in \{\ 0.1, 0.5, 0.9\ \}$.

4.3 Sampling from Potts Models

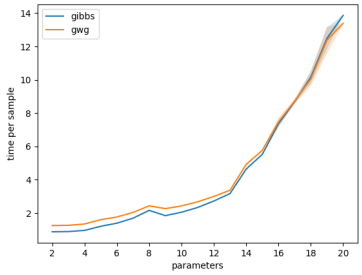
As mentioned in section 2.2.3, the Potts model is an extension of an Ising model which is categorical instead of binary. In this section we will investigate the effect of increasing the number of categories on the effectiveness of sampling. We sample from Potts models with 3 up to 10 categories (q).

The first thing to note is that Gibbs with Gradients scales better for higher values of q. In figure 6 we can see, for $J = 0.5$, that while the Gibbs sampler is initially faster, the Gibbs with Gradients sampler matches the speed at $q = 7$ and surpasses it at $q = 10$. Other values of J result in similar times. To

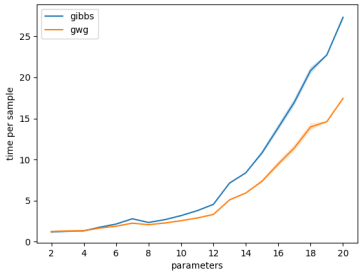
propose a new state the Gibbs sampler has to loop over all categories. This is faster at a low value of q, but quickly slows down as we increase the number of categories. On the other hand, Gibbs with Gradients is able to compute the gradient information with only a slight increase in computational cost. This gives it an advantage over the Gibbs sampler, making it able to achieve a lower time per sample.



(a) q = 4, J = 0.5



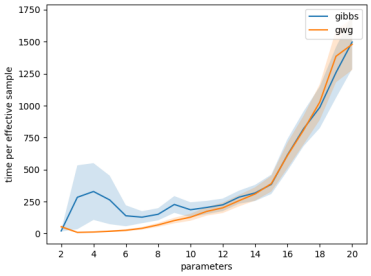
(b) q = 7, J = 0.5



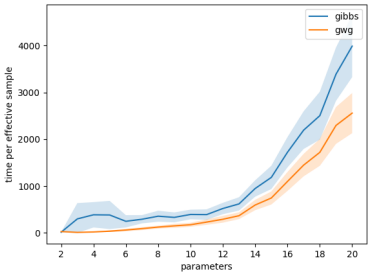
(c) q = 10, J = 0.5

Figure 6: The time per sample on a Potts model at 4, 7 and 10 categories with $J = 0.5$.

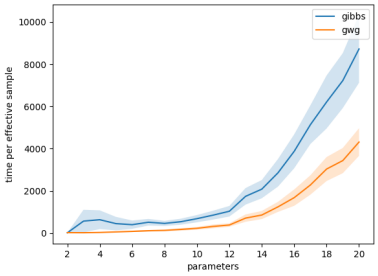
We can see the effect of the faster draws for Gibbs with Gradients in the time per effective sample as well. For nearly all values of q and J Gibbs with Gradients outperforms the Gibbs sampler. This advantage increases as the number of categories in the Potts model becomes larger. At $q = 10$ and $J = 0.5$ the Gibbs with Gradients sampler can draw an effective sample size at approximately twice the speed. Besides that, at lower values of q Gibbs with Gradients is just as fast at drawing effective samples as a Gibbs sampler. It is therefore a slightly better sampler for Potts models.



(a) q = 4, J = 0.5



(b) q = 4, J = 0.5



(c) q = 10, J = 0.5

Figure 7: The effect of the number of dimensions and number of categories on the time per effective sample size for various values of n_out and J.

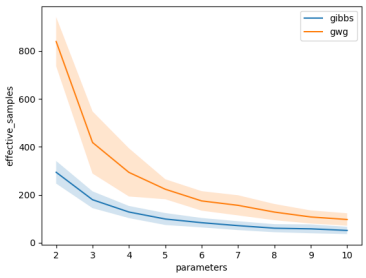
4.4 Sampling from Random Models

Randomly generated models allow us to increase the models to test the samplers against. We generate models as described in section 3. Some of these models might be invalid (section 3.4), therefore we will use rejection sampling to gather a set of models. We sample from a model, and if we find that a model produces an invalid output we will generate a new one.

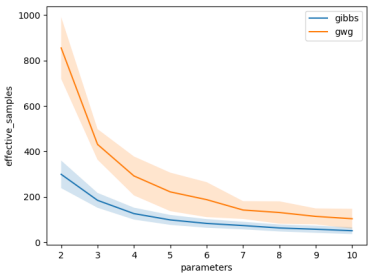
Using this method we generate modes with 2, 5, and 10 factors, up to 10 parameters and with $p = 0.25, 0.5$ and 0.75. p being the probability that a parameter is connected to a factor in the factor graph. For each combination of these values we generate 100 random models giving us a total of 8100 models. From these models we take 1000 draws each and compute the effective sample size, the time per effective sample and the time per sample.

Figures 8 and 9 show the effective sample size for sparse and dense models respectively. Similar to Ising and Potts models we see that the effective sample size is larger for the Gibbs with Gradients sampler. However, where the Gibbs sampler performs almost equally well on all 100 models for a given number of parameters, the effective sample size of the Gibbs with Gradients sampler has a much higher standard deviation. This is likely caused by models where the

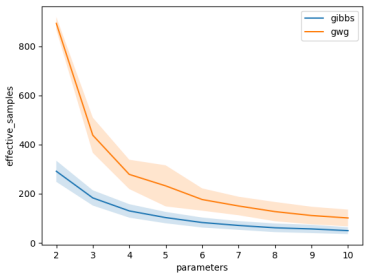
gradients are relatively uninformative due to their random nature.



(a) 2 factors

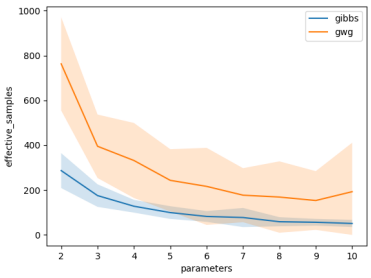


(b) 5 factors

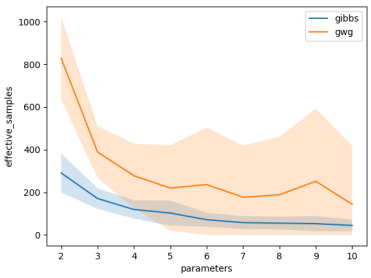


(c) 10 factors

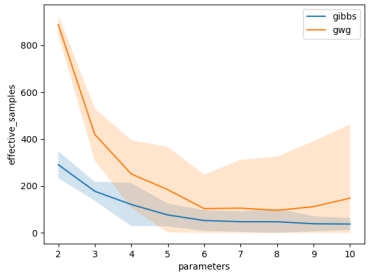
Figure 8: The effective sample size on sparse random models ($p = 0.25$) with 2, 5 and 10 factors.



(a) 2 factors



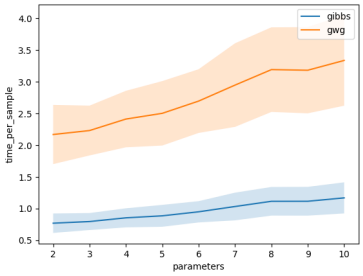
(b) 5 factors



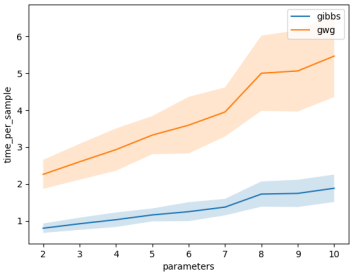
(c) 10 factors

Figure 9: The effective sample size on dense random models ($p = 0.75$) with 2, 5 and 10 factors.

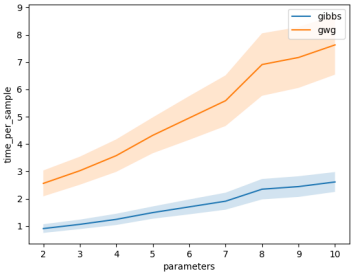
When we look at the time per sample (figure 10) we can see that for all values of p the time per sample increases in a similar fashion. It is no surprise that the time per sample slowly increases, as we generate each factor with a list of functions with length equal to the number of parameters. Increasing the p value results in a steeper increase in the time per sample. This can be explained by the fact that a higher p value will increase the expected number of parameters for each factor, thus making them computationally more expensive. Figure 10 shows the results for 5 factors, but similar results are found for 2 and 10 factors (see Appendix B). The Gibbs with Gradients sampler is about 2.5-4 times slower than the Gibbs sampler. This is a major impact on the sampling performance and would require a large increase in the effective sample size to be worth it.



(a) $p = 0.25$



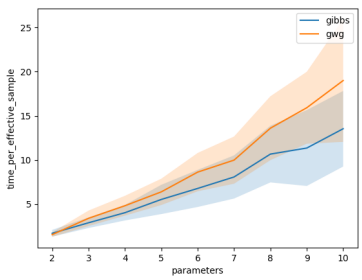
(b) $p = 0.5$



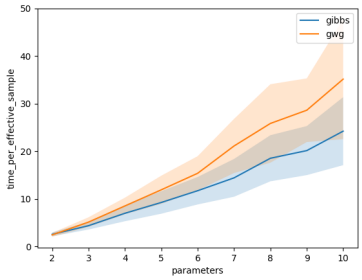
(c) $p = 0.75$

Figure 10: The time per sample on random models with 5 factors and varying factor densities.

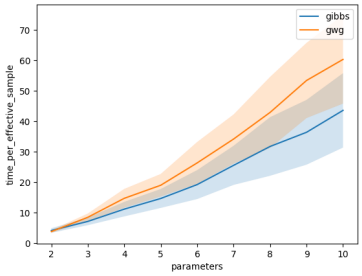
Indeed, if we measure the time per effective sample, the Gibbs with Gradients sampler is outperformed by the Gibbs sampler. Figure 11 show the time per effective sample for 2, 5, and 10 factors. While both samplers have a higher time per effective sample if we increase the number of factors, the Gibbs sampler always performs better than the Gibbs with Gradient sampler. For other values of p similar, and even more extreme, differences have been found. Always in the favour of the Gibbs sampler. The additional computational cost of Gibbs with Gradients has no additional benefits on our randomly generated models. For these models, the Gibbs sampler always outperforms Gibbs with Gradients with the additional benefit that it is a far simpler algorithm.



(a) 2 factors



(b) 5 factors



(c) 10 factors

Figure 11: The time per effective sample on random models with 2, 5, and 10 factors and $p = 0.25$.

5 Conclusions

We have analyzed the performance of samplers in terms of effective sample size, time per effective sample and time per sample. Using these metric, we compared two Markov Chain Monte Carlo samplers; the Gibbs sampler and the Gibbs with Gradients sampler. W. Grathwohl et al. [17] claimed that the latter outperforms the Gibbs sampler. We tested this claim against various models.

The first model we used was the Ising model, a model of ferromagnetism in statistical mechanics. Additionally, we sampled from Potts models, an extension of the Ising model with categorical values. Lastly, we generated random models where each factor in the factorization is a distribution from the exponential family and sampled from those.

Sampling from Ising models showed that, while Gibbs with Gradients manages to have a higher effective sample size, it failed to do this fast enough. The time per sample was significantly slower, resulting in a worse time per effective sample. This difference in sampling speed caused an increase in time per effective sample of up to three times. Therefore, it is faster to approximate expected values using the Gibbs sampler, making it the better choice for Ising models.

Potts models with few categorical values gave similar results, as that of the

Ising model. However, increasing the number of categorical values per parameter resulted in a significant decrease in the speed of the Gibbs sampler. Something by which the Gibbs with Gradients sampler was affected far less. Gibbs with Gradients seems to perform better on models with categorical parameters. The question remains whether this holds for other models with non-binary parameters.

Generating thousands of random models gave a more complete picture of the relative performance of the samplers. Gibbs with Gradients improves the effective sample size, as that was higher for nearly all tested randomly generated models. But, as we have seen with the Ising models, it lacks the sampling speed to truly compete with the Gibbs sampler. For random models, we see that the time per effective sample is again worse than that of the Gibbs sampler.

These experiments have allowed us to answer our research questions as stated in section 1:

How many draws can Gibbs with Gradients generate per second? How does this compare to other samplers?
For binary parameter discrete models, Gibbs with Gradients is a constant factor slower than the Gibbs sampler. Increasing the number of values for the parameter from binary to n categorical values, gives the advantage to Gibbs with Gradients for higher values of n.

How many effective samples can be drawn per second? Is Gibbs with Gradients able to draw more effective samples in the same amount of time?
Gibbs with Gradients is able to get a higher effective sample size. The slower sampling speed for models with binary parameters makes the effective sample size per second lower than that of the Gibbs sampler. For models with parameters with many categorical values both the sampling speed and the effective sample size are higher for Gibbs with Gradients and as a result the effective sample size per second is higher as well.

How does an increasing number of dimensions affect the sampling speed and effective sample size?
For both samplers the number of dimensions affects the effective sample size strongly. In Ising and Potts models, increasing the size of the sides of the lattice by one, severely decreases the effective sample size. In random models the effective sample size becomes less than 10% of the number of draws at as little as 10 dimensions.

How does Gibbs with Gradients compare to other discrete samplers, based on effective sample size, speed, and scalability to higher dimensions?
Gibbs with Gradients is clearly the better sampler when comparing the effective sample size to the Gibbs sampler, as is claimed by W. Grathwohl et al. [17]. However, due to the increased computational cost of the gradient information,

it does not have the sampling speed to give a practical advantage over the Gibbs sampler. Models with parameters with many categorical values might be the exception, but this would require further research.

6 Open questions

In section 4.3 we saw that Gibbs with Gradients could outperform the Gibbs sampler when the number of categorical values increased. One of the remaining open questions is whether these results will hold for more than 10 categorical values. And more generally, is there a class of models for which Gibbs with Gradients outperforms other state of the art samplers?

Secondly, section 4.4 discussed the results of sampling from random models. Due to the random nature of those models, it might have been the case that the gradients were simply too uninformative for Gibbs with Gradients. This raises a similar question as stated above, but more specifically it raises the question how the shape of the distribution affects Gibbs with Gradients. As Gibbs with Gradients makes of use the fact that most discrete distribution use an underlying continuous distribution, which is evaluated only at discrete points. These underlying continuous distributions do not have to be unique, as there can be infinitely many continuous distributions resulting in the same likelihood on some set of discrete values. Being able to effectively sample from a discrete distribution using gradient information relies heavily on these underlying continuous distributions. It would be interesting to see how Gibbs with Gradients would be effected by the choice of these distributions and how the gradients within those distribution affect the performance.

A more general problem facing these locally-informed samplers, is that they can easily get trapped in areas surrounded by low probability. If the Gibbs with Gradients sampler is in a state where each neighbor with hamming distance 1 has probability 0, it will never get out of that state. The same problem exists for the neighborhoods with hamming distances greater than 1, except that every neighbor of hamming distance up to some value k has to have a probability of 0. It might be that being locally-informed is not the best way to sample from models with discrete parameters. This problem is not unique to models with discrete parameters, as most samplers for models with continuous parameters can just as well get stuck in states surrounded by low probabilities. Solving this problem might benefit MCMC samplers, but might also prove to be very hard.

Lastly, a new sampler has been proposed by Sun et al. [43]. Their proposed Path Auxiliary Sampler (PAS) uses path auxiliary proposals, that is, proposals with hamming distance up to some value k. They claim that increasing the search space for the proposals can significantly improve Gibbs with Gradients. Besides this, they propose a faster version of their algorithm, named PAFS. As the main problem with the Gibbs with Gradients sampler is that the time per effective sample is usually worse than the Gibbs sampler, PAFS, or maybe even PAS, might provide us with a better alternative. Further research could compare PA(F)S to the Gibbs and Gibbs with Gradients samplers.

References

[1] Yves F. Atchadé. An adaptive version for the metropolis adjusted langevin algorithm with a truncated drift. *Methodology and Computing in applied Probability*, 8(2):235–254, 2006.

[2] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[3] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

[4] Miguel A. Carreira-Perpinan and Geoffrey Hinton. On contrastive divergence learning. In *International workshop on artificial intelligence and statistics*, pages 33–40. PMLR, 2005.

[5] Cyril Chimisov, Krzysztof Latuszynski, and Gareth Roberts. Adapting the gibbs sampler. *arXiv preprint arXiv:1801.09299*, 2018.

[6] J-J Daudin, Franck Picard, and Stéphane Robin. A mixture model for random graphs. *Statistics and computing*, 18(2):173–183, 2008.

[7] M.J. De Oliveira and Robert B. Griffiths. Lattice-gas model of multiple layer adsorption. *Surface Science*, 71(3):687–694, 1978.

[8] Simon Duane, Anthony D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

[9] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[10] Daniel Foreman-Mackey, David W. Hogg, Dustin Lang, and Jonathan Goodman. emcee: the mcmc hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306, 2013.

[11] Sylvia Fröhwirth-Schnatter and Sylvia Kaufmann. Model-based clustering of multiple time series. *Journal of Business & Economic Statistics*, 26(1):78–89, 2008.

[12] Alan E. Gelfand, Susan E. Hills, Amy Racine-Poon, and Adrian FM. Smith. Illustration of bayesian inference in normal data models using gibbs sampling. *Journal of the American Statistical Association*, 85(412):972–985, 1990.

[13] Charles J. Geyer. Introduction to markov chain monte carlo. *Handbook of markov chain monte carlo*, 20116022:45, 2011.

[14] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[15] Walter R. Gilks and Pascal Wild. Adaptive rejection sampling for gibbs sampling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 41(2):337–348, 1992.

[16] François Graner and James A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. *Physical review letters*, 69(13):2013, 1992.

[17] Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris J. Maddison. Oops i took a gradient: Scalable sampling for discrete distributions. *arXiv preprint arXiv:2102.04509*, 2021.

[18] Jun Han, Fan Ding, Xianglong Liu, Lorenzo Torresani, Jian Peng, and Qiang Liu. Stein variational inference for discrete distributions. In *International Conference on Artificial Intelligence and Statistics*, pages 4563–4572. PMLR, 2020.

[19] W. Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[20] Matthew D. Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[21] John Ingraham and Debora Marks. Variational inference for sparse and undirected models. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2017.

[22] Ernst Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift fur Physik*, 31(1):253–258, February 1925.

[23] Sorin Istrail. Statistical mechanics, three-dimensionality and np-completeness: I. universality of intracatability for the partition function of the ising model across non-planar surfaces. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 87–96, 2000.

[24] James Joyce. Bayes‘ Theorem. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

[25] John Lafferty, Andrew McCallum, and Fernando C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[26] Krzysztof Łatuszyński, Gareth O. Roberts, and Jeffrey S. Rosenthal. Adaptive gibbs samplers and related mcmc methods. *The Annals of Applied Probability*, 23(1):66–98, 2013.

[27] Andrew McCallum, Dayne Freitag, and Fernando C.N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Icml*, volume 17, pages 591–598, 2000.

[28] Geoffrey J. McLachlan, Richard W. Bean, and David Peel. A mixture model-based approach to the clustering of microarray expression data. *Bioinformatics*, 18(3):413–422, 2002.

[29] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[30] Radford M. Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[31] Akihiko Nishimura, David Dunson, and Jianfeng Lu. Discontinuous hamiltonian monte carlo for sampling discrete parameters. *arXiv preprint arXiv:1705.08510*, 2, 2017.

[32] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944.

[33] Taeyoung Park and Seunghan Lee. Improving the gibbs sampler. *Wiley Interdisciplinary Reviews: Computational Statistics*, page e1546, 2021.

[34] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[35] Judea Pearl. *Causality*. Cambridge university press, 2009.

[36] Edwin James George Pitman. Sufficient statistics and intrinsic accuracy. In *Mathematical Proceedings of the cambridge Philosophical society*, volume 32, pages 567–579. Cambridge University Press, 1936.

[37] Renfrey Burnard Potts. Some generalized order-disorder transformations. In *Mathematical proceedings of the cambridge philosophical society*, volume 48, pages 106–109. Cambridge University Press, 1952.

[38] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., 1993.

[39] D.A. Reynolds and R.C. Rose. Robust text-independent speaker identification using gaussian mixture speaker models. *IEEE Transactions on Speech and Audio Processing*, 3(1):72–83, 1995.

[40] Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

[41] Gareth O. Roberts and Jeffrey S. Rosenthal. General state space markov chains and mcmc algorithms. *Probability surveys*, 1:20–71, 2004.

[42] Elad Schneidman, Michael J. Berry, Ronen Segev, and William Bialek. Weak pairwise correlations imply strongly correlated network states in a neural population. *Nature*, 440(7087):1007–1012, 2006.

[43] Haoran Sun, Hanjun Dai, Wei Xia, and Arun Ramamurthy. Path auxiliary proposal for mcmc in discrete space. In *International Conference on Learning Representations*, 2021.

[44] Jianhua Yin and Jianyong Wang. A dirichlet multinomial mixture model-based approach for short text clustering. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 233–242, 2014.

[45] Giacomo Zanella. Informed proposals for local mcmc in discrete spaces. *Journal of the American Statistical Association*, 115(530):852–865, 2020.

A Ising model result graphs

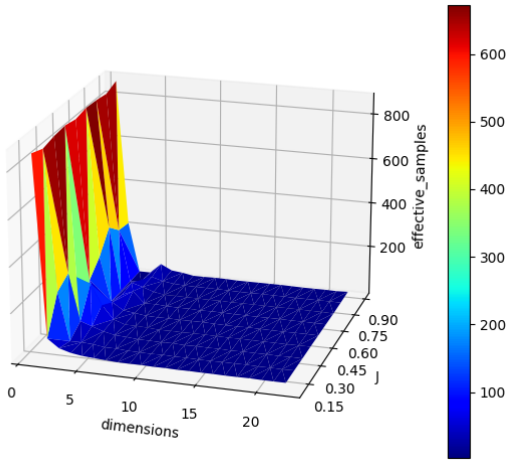


Figure 12: Gibbs Sampler: The effective sample size of an $N \times N$ lattice Ising model.

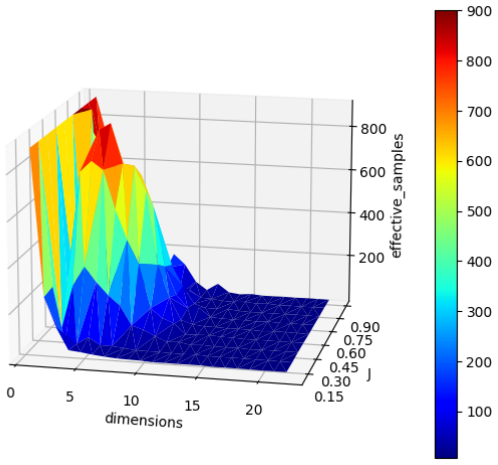
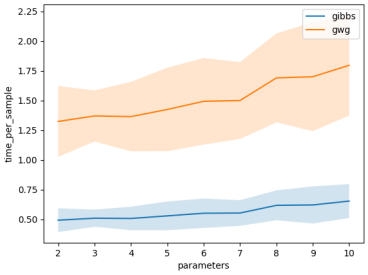
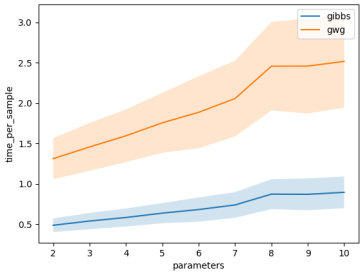


Figure 13: Gibbs with Gradients Sampler: The effective sample size of an $N \times N$ lattice Ising model.

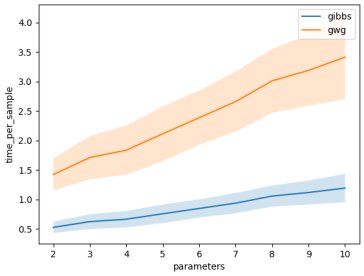
B Random model result graphs



(a) $p = 0.25$

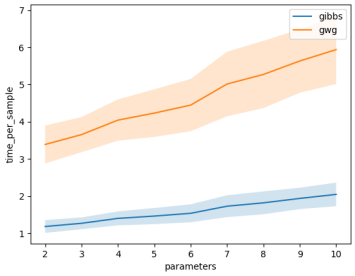


(b) $p = 0.5$

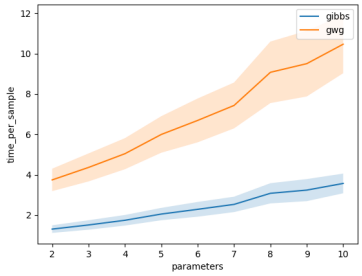


(c) $p = 0.75$

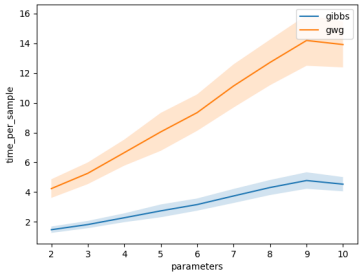
Figure 14: The effective sample size on random models with 2 factors and varying factor densities.



(a) $p = 0.25$



(b) $p = 0.5$



(c) $p = 0.75$

Figure 15: The effective sample size on random models with 10 factors and varying factor densities.