Perturb-and-max-product: Sampling and learning in discrete energy-based models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Perturb-and-MAP offers an elegant approach to approximately sample from an energy-based model (EBM) by computing the maximum-a-posteriori (MAP) configuration of a perturbed version of the model. Sampling in turn enables learning. However, this line of research has been hindered by the general intractability of the MAP computation. In fact, very few works venture outside tractable models, and when they do, they use linear-programming approaches, which as we will show, have several limitations. In this work, we present perturb-and-max-product (PMP), a parallel and scalable mechanism for sampling and learning in discrete EBMs. Models can be arbitrary as long as they are built using tractable factors. We show that (a) for Ising models, PMP is orders of magnitude faster than Gibbs and Gibbs-with-Gradients (GWG) at learning and generating samples of similar or better quality; (b) PMP is able to learn and sample from RBMs; (c) in a large, entangled graphical model in which Gibbs and GWG fail to mix, PMP succeeds.

1 Introduction

In this work, we will concern ourselves with the problem of black-box parameter estimation for a very general class of discrete energy-based models (EBMs), possibly with hidden variables. The golden measure of estimation quality is the likelihood of the parameters given the data, according to the model. However, the likelihood of EBMs is not computable in polynomial time due to the partition function , which in turn makes its optimization (parameter estimation) difficult.

This is an active research problem, with multiple approaches having been proposed over the last decades [35, 27, 31, 2, 12, 13, 11, 35, 16, 36, 18]. These methods typically fall in one or more of the following categories a) they depart from the maximum likelihood (ML) criterion to obtain a tractable objective, but in doing so they fail when the data does not match the model (e.g., pseudolikelihood [12]); b) they can only be applied to fully observed models (e.g., piecewise training [27]); c) they fail catastrophically for simple models (e.g., Bethe approximation [11]); d) they require additional designs (such as a custom encoder) for new models, i.e., they are not black-box (e.g, Advil [18]).

A general approach to discrete EBM learning that does not fall in any of those pitfalls is sampling: as we review in Section 2.1, sampling from an EBM results in efficient learning. Exact sampling is unfeasible in general, but approximate sampling works well enough in multiple models. Gibbs sampling is the most popular because it is simple, general, and can be applied in a black-box manner: the sampler can be generated directly from the model, even automatically [4]. Its main limitation is that for some models, mixing can be too slow to be usable. Recently the Gibbs-with-gradients (GWG) [7] sampler was proposed, which significantly improves mixing speed, enabling tractability of more models. A completely different approach to learning is Perturb-and-MAP [21], but this approach has not seen widespread adoption because the general MAP problem could not be solved fast enough or with good enough quality, see Section 2.4.1.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

In this work, we show that max-product, a relatively under-utilized algorithm, can be used to perform good enough MAP inference in many non-tractable cases very fast. Thus, perturb-and-max-product not only expands the applicability of perturb-and-MAP to new scenarios (we show that it can sample in highly entangled and multimodal EBMs in which the state-of-the-art GWG and traditional Perturb-and-MAP fail), but can also be much faster in practice than the state-of-the-art GWG.

2 Background

We review here ML learning, Perturb-and-MAP, belief propagation and how they connect.

2.1 Sampling and learning in discrete EBMs

Notation Consider an EBM with variables x described by a set of A factors $\{\phi_a(x_a;\theta_a) = \theta_a^\top \phi_a(x_a)\}_{a=1}^A$ and I unary terms $\{\phi_i(x_i;\theta_i) = \theta_i^\top \phi_i(x_i)\}_{i=1}^I$. Then x_a is the subset of variables used in factor a, θ_a is a vector of factor parameters and $\phi_a(x_a;\theta_a)$ is a vector of factor sufficient statistics. For the unary terms, the sufficient statistics $\phi_i(x_i;\theta_i)$ are the indicator functions, which are set to 1 when the variable takes a given value and to zero otherwise. Then the energy of the model can be expressed as $E(x,h) = -\sum_{a=1}^A \phi_a(x_a;\theta_a) - \sum_{i=1}^I \phi_i(x_i;\theta_i)$ or, collecting the parameters and sufficient statistics in corresponding vectors, $E(x) = -\Theta_a^\top \Phi_a(x) - \Theta_i^\top \Phi_i(x) = -\Theta^\top \Phi(x)$.

Given a discrete data set $\mathcal{D} \equiv \{x^{(n)}\}_{i=1}^N$, where $x \in \{0,1\}^D$, the log-likelihood of the parameters for a generic EBM in which some variables (x_h) are hidden can be expressed as

$$\mathcal{L}(\Theta|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^N \log p(x^{(n)};\Theta) = \frac{1}{N}\sum_{n=1}^N \log \sum_{x_h} \exp(\Theta^\top \Phi(x^{(n)}, x_h)) - \log \sum_x \exp(\Theta^\top \Phi(x)) \quad (1)$$

The learning problem can be described as efficiently maximizing the key quantity Eq. (1). However, Eq. (1) can only be computed for small models, due to the summation over x (the partition function), which involves an exponential number of summands. In order to use gradient ascent methods, we need the gradient of Eq. (1) w.r.t. Θ, which is $\nabla_\Theta \mathcal{L}(\Theta|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^N \langle \Phi(x^{(n)}, x_h)\rangle_{p(x_h|x^{(n)};\Theta)} - \langle \Phi(x)\rangle_{p(x;\Theta)}$. From this gradient we can make two standard observations: First, at a maximum of Eq. (1), the expectation of the sufficient statistics according to the data and the model match, since the gradient needs to be zero. Second, sampling enables efficient learning: we can efficiently approximate the expected sufficient statistics, with no bias, by sampling from the model (both unconditionally and conditioned on $x^{(n)}$). Exact sampling is also unfeasible, so we turn to approximate sampling.

2.2 Perturb-and-MAP

In [21], a family of samplers, including the following convenient form, was proposed:

$$x^{(s)} = \arg\max_x (\Theta_a^\top \Phi_a(x) + (\Theta_i + \varepsilon^{(s)})^\top \Phi_i(x)) \quad (2)$$

where $\varepsilon^{(s)}$ is a random perturbation vector with as many entries as variables in the model. This means that to generate a sample, first we draw $\varepsilon^{(s)}$ from some distribution, use it to perturb the unary terms, and then we solve a standard maximum a posteriori (MAP) problem on the perturbed model.

While MAP inference is NP-complete, it is easier than partition function computation which is #P-complete. In fact, in log-supermodular models, such as attractive Ising models, the exact MAP can be computed using graph-cuts, but the partition function still needs exponential time.

The distribution of the samples $\{x^{(s)}\}$ will of course depend on the perturbation density. In models with only unary terms, we can get the Gibbs distribution $p(x^{(s)}) \propto \exp(\Theta^\top \Phi(x^{(s)}))$ using the zero-mean density $\varepsilon \sim \mathrm{Gumbel}(\varepsilon; -c, 1)$, where $c \approx 0.5772$ is the Euler-Mascheroni constant (see Lemma 1, [21]). For the more realistic case of models containing non-unary terms, the density of the samples will *not* match the Gibbs density, but it will be an approximation.

The Perturb-and-MAP approach also allows for the computation of an upper bound on the partition function, $\log \sum_x \exp(\Theta^\top \Phi(x)) \leq \langle \max_x (\Theta_a^\top \Phi_a(x) + (\Theta_i + \varepsilon^{(s)})^\top \Phi_i(x))\rangle_{p(\varepsilon)}$, (see Corollary

1, [9]). If we use this approximation in the log-likelihood (1), we get a much easier target for optimization

$$\hat{\mathcal{L}}(\Theta|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N} \Theta^\top \Phi(x^{(n)}) - \langle \max_x (\Theta_a^\top \Phi_a(x) + (\Theta_i + \varepsilon^{(s)})^\top \Phi_i(x)) \rangle_{p(\varepsilon)} \tag{3}$$

whose gradient is $\nabla_\Theta \hat{\mathcal{L}}(\Theta|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N} \Phi(x^{(n)}) - \langle \Phi(x^{(s)}) \rangle_{\hat{p}(x^{(s)};\Theta)}$. Note that this gradient has the same form as the gradient of the true likelihood, but with the exact sampler replaced by the approximate one. So this can be thought of as the approximate gradient of the true log-likelihood, or as the true gradient of the approximate log-likelihood.

The expectations in Eq. (3) and its gradient can be easily approximated, without bias, using the sample mean. Thus, we can use stochastic gradient descent (SGD) and minibatches, to get a generally applicable technique for graphical models *if* we can solve the MAP problem efficiently. So far, the application of perturb-and-MAP has been restricted to models whose structure allowed exact MAP solutions or allowed LP relaxations of the MAP problem to produce accurate results. As we will show in Section 5.2, even simple Ising models violate these assumptions.

2.3 The Bethe approximation and belief propagation

We take an apparent detour to talk about the Bethe approximation and belief propagation. We will tie this back to MAP inference in the next subsection.

The Bethe free energy uses variational arguments to approximate the partition function. Using an arbitrary variational distribution $q(x)$ we can lower bound the partition function. At the maximum with respect to this distribution, we obtain equality with the partition function:

$$\beta^{-1} \log \sum_x \exp(\beta \Theta^\top \Phi(x)) = \max_{q(x)} \left\{ \langle \Theta^\top \Phi(x) \rangle_{q(x)} + \beta^{-1} \mathcal{H}(q(x)) \right\} \tag{4}$$

where we keep an *inverse temperature* parameter β throughout the derivation. Obviously, the maximization is still intractable. Instead, the Bethe approximation to the partition function [37] is

$$\begin{aligned} &\max_{q_a(x_a), q_i(x_i)} \sum_{a=1}^{A} \langle \phi_a(x_a;\theta_a) \rangle_{q_a(x_a)} + \sum_{i=1}^{I} \langle \phi_i(x_i;\theta_i) \rangle_{q_i(x_i)} + \beta^{-1} \mathcal{H}_B(q_a(x_a), q_i(x_i)) \\ &\text{s.t.} \sum_{x_{a\setminus i}} q_a(x_a) = q_i(x_i), \quad \sum_{x_i} q_i(x_i) = 1, \text{ and } q_a(x_a) \geq 0, \end{aligned} \tag{5}$$

where two approximations have been made. The Bethe free entropy $\mathcal{H}_B(q_a(x_a), q_i(x_i)) = \sum_a \mathcal{H}(q_a(x_a)) + \sum_i (1 - f_i)\mathcal{H}(q_i(x_i))$ (exact for trees, approximate for loopy models) is used instead and, since we only need the marginals of $q(x)$, we only enforce them to be locally consistent. These are called pseudo-marginals, since they might not correspond to any joint density.

To convert this into an unconstrained optimization problem, we move to the dual. The dual variables are called "messages", since there is one of them associated to each factor-variable pair. See [38] for details. Setting the gradient to 0, we recover the belief propagation fixed point equations

$$\begin{aligned} m_{i\to a}(x_i) &= \phi_i(x_i;\theta_i) + \sum_{b \in \mathrm{nb}(i)\setminus a} m_{b\to i}(x_i) \\ m_{a\to i}(x_i) &= \beta^{-1} \log \sum_{x_{k\setminus i}} e^{\beta\left(\phi_a(x_a;\theta_a) + \sum_{k\in\mathrm{nb}(a)\setminus i} m_{k\to a}(x_k)\right)}, \end{aligned}$$

where $\mathrm{nb}(\cdot)$ denotes the neighbors of a factor or variable. Updates must proceed sequentially factor by factor. Convergence to a fixed point is not guaranteed for loopy models. We can recover the primal quantities from the messages. Among them, the unary marginals: $q_i(x_i) \propto \exp(\beta(\phi_i(x_i;\theta_i) + \sum_{b\in\mathrm{nb}(i)} m_{b\to i}(x_i))$. The recovered primal value in Eq. (5), for $\beta = 1$, is an approximation to the log-partition function, but cannot be used for learning in general, see [22, 11], and Section 5.1.

2.4 Max-product message passing

We are interested in the limit $\beta \to \infty$ (zero temperature). The l.h.s. of Eq. (4), i.e., the quantity that we are approximating, becomes $\max_x \Theta^\top \Phi(x)$, exactly the MAP problem. The update for $m_{i\to a}(x_i)$ remains unchanged, the other simplifies to $m_{a\to i}(x_i) = \max_{x_{k\setminus i}} \phi_a(x_a;\theta_a) + \sum_{k\in \text{nb}(a)\setminus i} m_{k\to a}(x_k)$. The new "unary marginals" $x_i = \arg\max_c \phi_i(x_i = c;\theta_i) + \sum_{b\in\text{nb}(i)} m_{b\to i}(x_i = c)$ provide the maximizing configuration. This algorithm is known as max-product or belief revision. A damping factor $0 < \alpha \leq 1$ in the message updates can be used without affecting the fixed points but improving convergence. In fact, even parallel updates often provide satisfactory results with as little as $\alpha = \frac{1}{2}$.

2.4.1 The connection with linear programming

When taking the limit $\beta \to \infty$, the astute reader will have probably noticed that, rather than modifying the fixed point updates, we could have removed the entropy in the r.h.s. of Eq. (5) and obtained a simple linear program (LP). Indeed, this is possible, and there is a vast literature on how to tackle the resulting LP (a continuous relaxation of the MAP problem) [6, 20, 15, 33, 19, 14, 32]. In fact, most approximate MAP inference research falls in this category, exploiting the convenience of the convexity of LPs.

It is common, although incorrect, to then assume that taking the limit $\beta \to \infty$ in the fixed point equations, i.e., max-product, is just some sort of coordinate-wise dual algorithm to optimize the same LP. This is not the case. The connection between the LP and max-product has been studied in detail in [34]. Max-product is a minimax algorithm in the dual, and has different fixed points than the LP.

Here we argue that, for the purpose of sampling, max-product is actually superior to LP-based solvers. The perturbed MAP problems are highly multimodal, even for simple cases such as Ising models. The Bethe free energy is a non-convex function with multiple minima. As $\beta \to \infty$, all those minima flatten out, but still exist [34], and become the fixed points of max-product, so the multimodality of the problem is preserved, and hopefully one of the modes reached. In contrast, the LP approach merges all those minima into a single point or flat region, the global minimum, and the structure is lost. The relaxed solution of the LP on simple Ising models is often 0.5 for all or most variables, a result that cannot be rounded to any useful discrete answer. We test this empirically in Section 5.2.

3 Perturb-and-max-product (PMP) for sampling and learning

As explained in Section 2.4.1, LP-based solvers are not well-suited for perturb-and-MAP inference, and will fail even in simple cases. Furthermore, max-product is an anytime algorithm whose precision can be controlled by the number of times that the messages are updated, much like the update steps of Gibbs sampling, enabling an easy trade-off between compute time and accuracy.

Perturb-and-max-product (PMP) learning is presented in Algorithm 1. Minibatched PMP samples are used to estimate the gradient of the approximate likelihood Eq. (3). The procedure is highly parallel: at each time step t we can compute the updates for all the factors (and for all the samples in the minibatch) at the same time. All the computation is local in the graph: the message updates only involve messages arriving to the factor. In the algorithm, we use $\Theta|_{x^{(n)}}$ to refer to the modification of the unary terms in Θ that clamps the visible variables to the observed values $x^{(n)}$. The damping α is set to $\frac{1}{2}$, since this value seems to offer a good trade off between accuracy and speed in most cases.

A natural benchmark for this general-purpose algorithm is Gibbs sampling (GS). Both GS and PMP have a computational complexity that scales as $\mathcal{O}(T\sum_{i=1}^{I} f_i)$, where f_i is the number of factors variable i is connected to, and T is the number of times we do a full sweep over all the variables. Sequentially, PMP is a constant factor slower than GS, given the more involved updates. However, the high parallelism of PMP makes it up to orders of magnitude faster in practice. Recently, a strong competitor to GS was presented: Gibbs with gradients (GWG) [7]. GWG essentially optimizes the order in which variables in the model are updated, thus providing faster mixing for the same number of updates (and with 2-3$\times$ times more compute), so we will also consider it in our comparisons.

Intuitively, Gibbs sampling propagates less information than PMP, since rather than real-valued messages, it only propagates discrete values. To illustrate this effect, consider a simple chain EBM: Gibbs sampling can be exponentially slow [1], whereas PMP will find the optimal configuration in linear time. The trade-off is that PMP is not unbiased (even asymptotically) and a bit less general

than GS: it can only be applied when all the factors in the model are max-marginalizable. Fortunately, most small factors and some not-so-small ones[1] are, and they can be combined without limitation into rich EBMs.

One important distinction when using PMP for learning is not to confuse the learned $\hat{\Theta}_{\text{PMP}}$ (which are the parameters that the PMP sampler will use to try to match to the data distribution) with the parameters of a Gibbs distribution. As we show in Section 5.1, the learned $\hat{\Theta}_{\text{PMP}}$ can be very far from the Gibbs distribution parameterization that produces the same density.

Algorithm 1: Learning and sampling with perturb-and-max-product (PMP)

Input: Sufficient statistic $\Phi(\cdot)$, data $\{x^{(n)}\}_{n=1}^{N}$, step size η, minibatch size S, sampling steps T
Output: A learned set of parameters Θ

```
Θ ~ N(0, 0.01)                                                    // Init
for l ← 1 to L do
    for s ← 1 to S do
        n ← Uniform(1, N)                          // Choose one sample from the training set
        y_+^(s) ← pmp(Θ|_{x^(n)}, T)                          // Get a sample from the posterior
        y_-^(s) ← pmp(Θ, T)                                       // Get a sample from the prior
    end
    Θ ← Θ + η( (1/S) Σ_{s=1}^{S} Φ(y_+^(s)) − (1/S) Σ_{s=1}^{S} Φ(y_-^(s)) )     // Stochastic gradient ascent
end
return Θ
Procedure pmp(Θ, T)
    ε_i ~ Gumbel(−c, 1) ∀_i; m^(0)_{i→a}(x_i) ← 0 ∀_{a,i,x_i}; m^(0)_{a→i}(x_i) ← 0 ∀_{a,i,x_i}      // Init
    for t ← 1 to T do
        m^(t)_{i→a}(x_i) ← φ_i(x_i; θ_i + ε_i) + Σ_{b∈nb(i)\a} m^(t−1)_{b→i}(x_i) ∀_{a,i,x_i}        // Max-product
        m^(t)_{a→i}(x_i) ← (1/2) m^(t−1)_{a→i}(x_i) + (1/2) max_{x_{k\i}} ( φ_a(x_a; θ_a) + Σ_{k∈nb(a)\i} m^(t)_{k→a}(x_k) ) ∀_{a,i,x_i}
    end
    x_i ← arg max_c φ_i(x_i = c; θ_i + ε_i) + Σ_{b∈nb(i)} m_{b→i}(x_i = c) ∀_i                  // Decode
    return x
```

4 Related work

Learning using Perturb-and-MAP has been proposed in different variants in recent literature, [21, 9, 5, 25, 24, 10, 24], with the common theme being that either only tractable MAP inference was considered or LP solvers (including graph-cuts in this category) were used. As we argued in Section 2.4.1, this will not work on multimodal problems (such as those invariant to permutations like in Section 5.6) and will perform poorly even in simple settings, as demonstrated in Section 5.2.

In the seminal work [21], the authors suggest using QPBO [23] for the MAP problem, which, as a convex method, also fails even in small Ising models. The approaches in [9, 10] use the max-product LP (MPLP) [6] to solve the MAP problem, whose poor performance in non-attractive problems we show in Section 5.2. The recent work [24] is limited to tractable models. The structured learning of [25] also relies on log-supermodular potentials that make MAP inference tractable.

A different approach can be found in [5]: it absorbs the parameters of the model in the perturbation distribution, and projects each data point to a concrete perturbation value (the MAP of the perturbation given the data point). Then the parameters of the perturbation are slightly adjusted towards fitting the projected data, and the process is repeated, in the form of hard expectation maximization. The optimizations are "inverse" problems in the continuous space of perturbations, typically solved using quadratic programming. It has the theoretical advantage of being able to handle any noise distribution, but it is very computationally demanding and does not support Gumbel-distributed perturbations.

[1] For instance, n-wise factors require $\mathcal{O}(C^n)$ compute (where C is the cardinality of the variables), logical factors (such as the OR of n variables) only need $\mathcal{O}(n)$ time [19], cardinality potentials limiting k out of n variables to be on need $\mathcal{O}(kn)$ [28], other higher-order potentials [29], etc.

5 Experiments

We now study the empirical performance of PMP for sampling and learning. We will measure performance using the maximum mean discrepancy (MMD) [8], with the standard average Hamming distance kernel $\exp \frac{-\sum_{d=1}^{D}[x_d \neq x'_d]}{D}$. We will use the term "full sweep" to refer to a full MCMC update for all variables in GS and GWG, and a single update of all the messages for PMP. Experiments were run on a GeForce RTX 2080 Ti. All compared methods are coded on JAX [3] and run with changes only in the samplers for maximum comparability of their runtimes.

5.1 Learning and sampling from the "wrong" model

We want to clarify that during learning PMP selects the parameters $\hat{\Theta}_{\text{PMP}}$ so as to match the expected sufficient statistics of the data, when *sampling*. Those are *not* the parameters of a Gibbs distribution. To drive this point home, and also to show that PMP can learn and sample from a distribution that cannot be learned with the Bethe approximation, we use the toy model from [22].

Consider an EBM with 4 variables, fully connected, with energy function $E(x) = -\theta \sum_{i<j \in \text{edge}} x_i, x_j$, where $x \in \{-1, +1\}^4$. This model is not learnable using the Bethe approximation for $\theta > 0.316$ [22]. We set $\theta = 0.5$ and run PMP learning on infinite data sampled from this model (this is feasible, since the expected statistics can be computed exactly). We obtain $\hat{\theta}_{\text{PMP}} \approx 0.325$. We might be tempted to compare this value with the original $\theta = 0.5$ of the generating Gibbs distribution to assess the quality of the PMP approximation, but that would be incorrect. $\hat{\theta}_{\text{PMP}}$ is the parameter of the PMP sampler, not of a Gibbs distribution. To see this, note that the KL divergence from data to i) the Gibbs distribution $\text{KL}(p(x|\theta = 0.5)||p(x|\theta = 0.325)) = 0.129$, is much worse than to ii) the learned sampling distribution $\text{KL}(p(x|\theta = 0.5)||p_{\text{PMP}}(x|\hat{\theta}_{\text{PMP}} = 0.325)) \approx 0.007$, which produces an almost perfect match. One way to look at this is that we are learning the "wrong" model, but it is wrong in such a way that it compensates for the way in which PMP sampling is also inexact, thus resulting in a learned sampling distribution that is closer to that of the data.

5.2 MPLP: Max-product is not broken

In the seminal paper "Fixing max-product" [6], Globerson et al. show how the LP Eq. (5) with $\beta \to \infty$ can be solved via a convergent sequence of message passing updates, which they call max-product LP (MPLP). We show that MPLP can perform very poorly even in simple cases. The first problem will be an Ising model. We generate 100 EBMs with energy $E(x) = -\sum_{i,j \in \text{edge}} w_{ij} x_i, x_j - \sum_i b_i x_i$ where w_{ij} is uniformly sampled in $[-2, 2]$ and b_i in $[-0.1, 0.1]$, with $x \in \{-1, +1\}$. We consider dimensions $\{5, 10, 15\}$. Then second problem, a 2D cyclical square lattice with energy $E(x) = -0.1 \sum_{i,j \in \text{edge}} x_i, x_j$, with side sizes of $\{5, 10, 15\}$. For both problems, we calculate the exact partition function and estimations using PMP and MPLP (see Section 3). The same perturbations are used for both methods, so they solve the same exact MAP problem. These are upper bounds, so the approximation error should be positive. We run PMP for 200 full sweeps, and MPLP until the message updates change by less 10^{-6} (more than 200 full sweeps). In Figure 1[left] we see that this is the case for the 2D lattice, where both approaches perform similarly. For Ising models, neither method is an actual upper bound (since the $\max$ and mean are only computed approximately), but MPLP is significantly worse, see Figure 1[center] . Furthermore, MPLP is a sequential algorithm (for which the damping trick does not work in practice), so for the 15×15 2D lattice sampling from MPLP used $500\times$ more compute time than PMP, making it impractical for sampling.

5.3 Ising model: MNIST zeros

Here we train a fully connected Ising model on contour-detected zeros from MNIST [17] (top row of Figure 4[left]). For learning we use ADAM with 0.001 learning rate, 100 chains, 50 full sweeps. The MCMC chains are persistent, and a non-persistent version which resets on every learning iteration is marked with the suffix "reset". After training, we produce new samples following each model and method for a varying number of sampling steps. See Figure 1[right].

As expected, GWG outperforms Gibbs. The reset versions are truncated, biased samplers, during learning, so they perform better when sampling for a similar number of full sweeps. Despite the

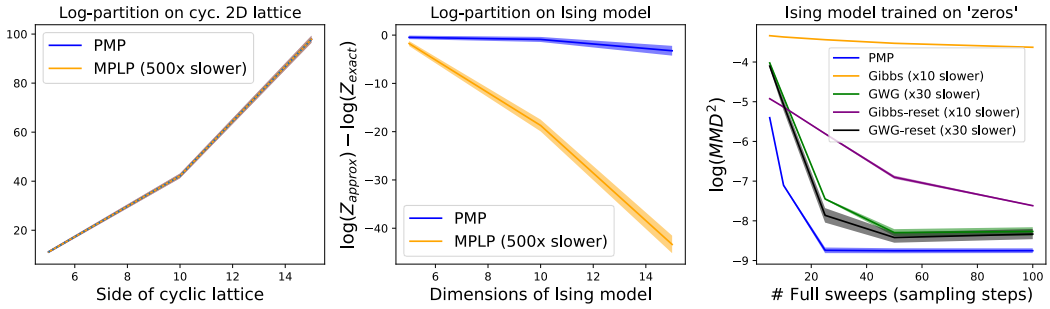

Figure 1: [Left and center] Log-partition estimation error using PMP and MPLP. Closer to zero is better. Shaded area represents ± 1 std deviation. [Right] Sampling quality for different sampling methods, keeping training fixed and varying the number of full sweeps at sampling time.

prevalence of persistent contrastive divergence (PCD), short-run reset samplers seem best when we want to generate a high quality sample with a truncated MCMC chain. PMP outperforms all the other methods, and does so with a much smaller compute time, as indicated in the legend. PMP is actually performing around $3\times$ more operations than Gibbs, but its parallelism more than makes up for it.

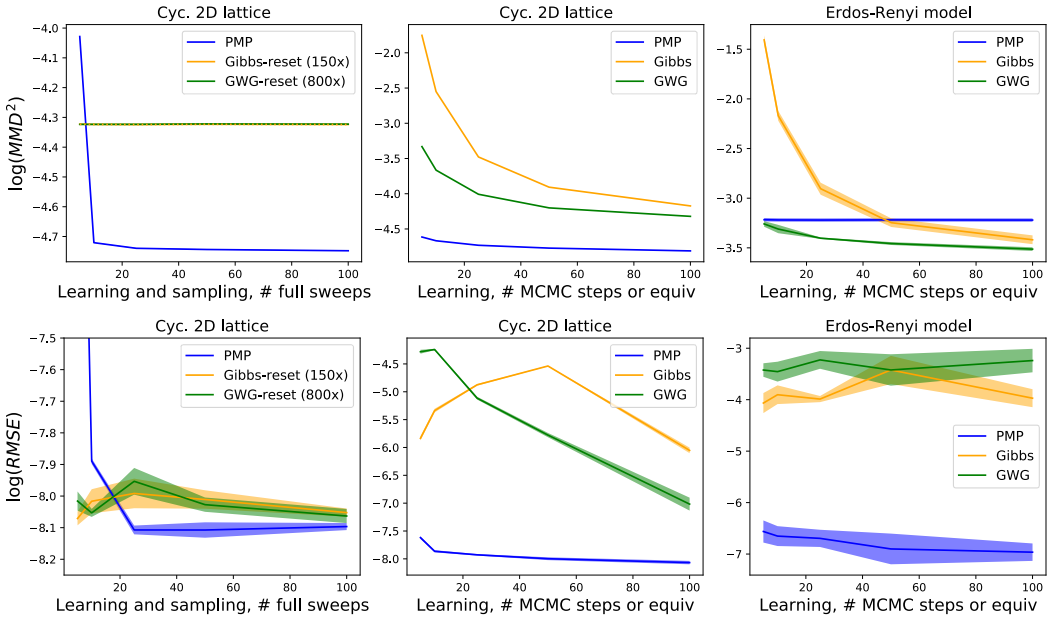

Figure 2: Performance on the lattice model ($\theta = -0.1$, side 25) and the Erdös-Renyi graphs from [7]. [Left column] x-axis determines the number of full sweeps for training and sampling, for all methods. [Center and right column] x-axis determines the number of *individual variable updates* for Gibbs and GWG at training time. PMP is trained for a number of full sweeps that result in less training time than GWG at the same x-axis location. To sample, all methods are run for 50 full sweeps.

5.4 Structured models: 2D lattices and Erdös-Renyi graphs

Here we reproduce the experiments from [7] with structured models, following precisely their definition, sampling, and learning, see details in Appendix A.2. Their performance metric is the root mean square error (RMSE) of the estimated parameters (which we include for reference), but that metric is not as meaningful for PMP, since as we saw in Section 5.1, a large difference in the parameters does not imply low quality samples.

Non-persistent training In the left column of Figure 2 we show the performance of training in a 2D lattice from [7] with 20 chains and a varying number of full sweeps both for learning and sampling. The chains are reset in between learning iterations. All methods perform well, with PMP taking the lead (even in RMSE), and also being substantially faster (see legend).

Persistent training This is the actual setting from [7], shown in the remaining plots of Figure 2. Less than a full sweep is run in between iterations and chains are not reset. Samples are produced by a fixed 50 full sweeps for all methods. The x-axis measures here the number of *individual variable updates* in between learning iterations, so all the points in the x-axis correspond to less than a full sweep. A persistent version of PMP is outside the scope of this work (although we provide a possible approach in Appendix A.1). Just as the non-persistent MCMC samplers, non-persistent PMP cannot produce meaningful samples in less than a full sweep, so it cannot compete by that metric. Instead, we adjust the number of full sweeps of PMP to take equal or smaller compute time than GWG at the same x-axis location. This allows a few full sweeps of PMP, sufficient to significantly outperform GWG with the same compute budget. Also, notice that once again the persistent regime is not the most conducive to highest sampling quality, for any of the methods tested, so despite being faster and more commonly used, it might not be the best choice for all applications.

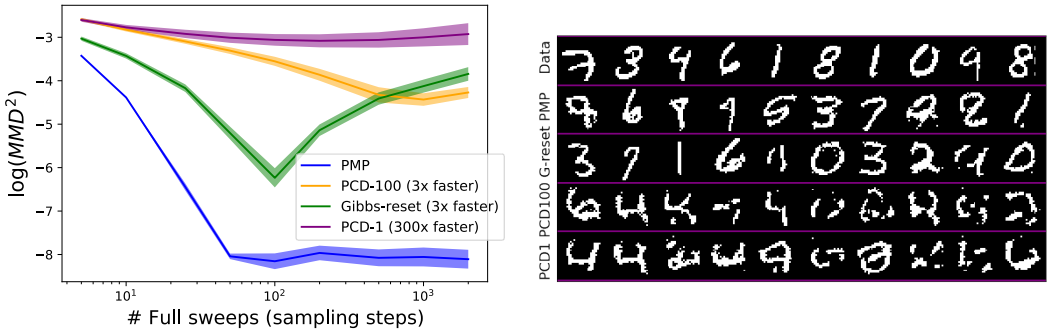

Figure 3: Left: Sampling quality for an RBM trained once and sampled for an increasing number of full sweeps. Right: Data and samples per method at 100 full sweeps. More samples in Appendix A.2.

5.5 Restricted Boltzmann machines, learning and sampling

Here we train an RBM on MNIST data and sample from it using PMP. To the best of our knowledge, this is the first reported instance of a perturb-and-MAP method able to do this. Indeed, we ran the author's code from [30] and the samples did not look like MNIST by a far margin. No samples are reported in the paper. We were able to reproduce their AIS metrics. See Appendix A.3 for discussion.

We train a restricted Boltzmann machine (RBM) with 500 hidden units on MNIST using 100 non-persistent full sweeps of PMP and block Gibbs (Gibbs-reset) and also single-sweep (PCD-1) and 100-sweep (PCD-100) persistent block Gibbs variants. We use stochastic gradient descent with learning rate 0.01 and train for 200 epochs using minibatches of size 100.

Note that here it does not make sense to use GWG, since the block Gibbs sampler is much more efficient than a black-box selection of the next variable to sample. The block Gibbs sampler is informed by the structure of the model. However, note that PMP is not. It is processing all factors in parallel, unaware of the conditional independences in the RBM.

After learning, the MMD for each model as the number of full sweeps increases is reported in Figure 3[left]. It is interesting to see the marked peak of Gibbs-reset at the 100 full sweeps mark. Indeed, we have learned a biased sampler tuned for 100 full sweeps, so more MCMC steps make that model *worse*. Again, PMP outperforms the competition in sample quality. However, in this particular case, the block Gibbs sampler specifically designed for RBMs can leverage as much parallelism as PMP, and having a smaller multiplicative constant due to its simplicity, it is faster than PMP. The trade-off is not being black-box and providing worse MMD figures.

Figure 3[right] shows sampling results for each method after 100 full sweeps. Samples for other numbers of full sweeps and a brief discussion can be found in Appendix A.2.

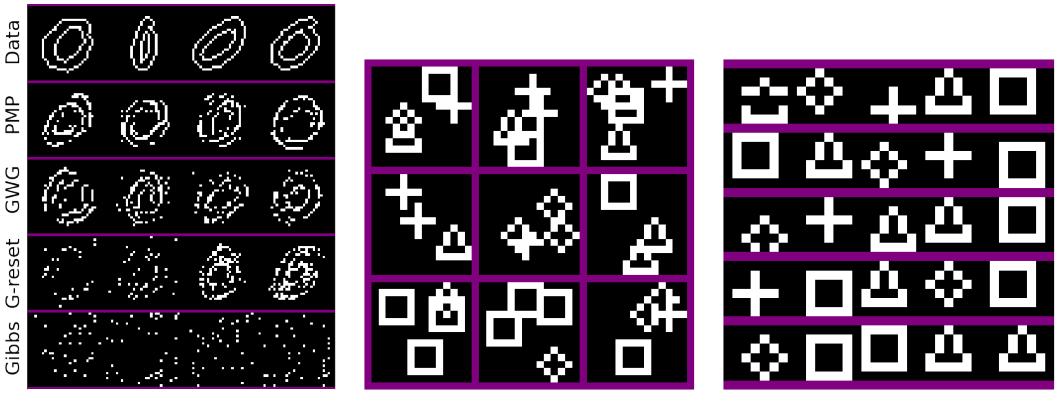

Figure 4: Left: Data and samples from MNIST zeros Ising model after 25 full sweeps of sampling using each tested method. Center: 9 examples of blind deconvolution dataset (full dataset X in the Appendix A.2). Right: 5 samples of the posterior using PMP.

5.6 2D blind deconvolution

Finally we consider the problem of sampling from an EBM with 40180 binary variables that are intricately connected, and which are invariant to permutations.

The generative model combines two sets of variables S, and W to form a set of images X. W (size: $n_{\text{feat}} \times \text{feat}_{\text{height}} \times \text{feat}_{\text{width}}$) contains several 2D binary features. S (size: $n_{\text{images}} \times n_{\text{feat}} \times h \times w$) is a set of binary indicator variables representing whether each feature is present at each possible image location. The two are combined by convolution, placing the features defined by W at the locations specified by S in order to form an image. The image is binary, so feature overlap acts as an OR, not a sum. All entries in S and W have independent Bernoulli prior. Given 100 images, we aim at recovering S and W. This is done by clamping X and sampling from the posterior. Observe that, unlike S, W is shared by all images. This makes all variables highly interdependent and the posterior highly multimodal. Also, note that permuting the first dimension of W and the second dimension of S in the same manner does not change X, so this naturally results in multiple equivalent modes.

This is essentially a Bayesian version of the problem in [26], where the weights have been converted into binary variables. This problem contains Boolean matrix factorization as a particular case.

Figure 4 shows in the center 9 images from the data set X, and on the right 5 samples (one per row) from the posterior as obtained by PMP after 1000 full sweeps. Each sample comes from a consecutive random seed, showing that the model is consistently good at finding samples from the posterior. The W used for inference did not match the one used for generation: we added one extra feature and made all of them slightly bigger, since in a real scenario we are not expected to know the ground truth generative parameters. The samples from the posterior used that additional feature to include an extra copy of the symbols used for generation, but otherwise looked correct. GWG was much slower per full sweep and we were not able to produce any meaningful posterior samples in a reasonable time. GWG did work when clamped S or W, but not when trying to sample from both simultaneously. Full training data is in Appendix A.2.

6 Discussion

We have presented PMP, a method for sampling and learning in EBMs. It improves over standard Perturb-and-MAP in that it offers a MAP solver that is fast, parallel, anytime, and produces high quality solutions in both simple and challenging problems in which LP-based MAP solvers fail.

Compared with Gibbs sampling and GWG, PMP offers faster mixing for a given number of full sweeps, and its parallelism helps it complete those full sweeps in substantially less time. It also has the ability to solve some problems exponentially faster than Gibbs sampling, theoretically in the case of chains and experimentally in cases like Section 5.6. Its main limitation is that it is only applicable in EBMs consisting of max-marginalizable factors, although this class is very rich, contains many well-known models, and can be made arbitrarily complex.

References

[1] Adrian Barbu and Song-Chun Zhu. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1239–1253, 2005.

[2] Julian Besag. Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 24(3):179–195, 1975.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[4] Sarah Depaoli, James P Clifton, and Patrice R Cobb. Just another gibbs sampler (jags) flexible software for mcmc implementation. *Journal of Educational and Behavioral Statistics*, 41(6):628–649, 2016.

[5] Andreea Gane, Tamir Hazan, and Tommi Jaakkola. Learning with maximum a-posteriori perturbation models. In *Artificial Intelligence and Statistics*, pages 247–256. PMLR, 2014.

[6] Amir Globerson and Tommi Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. *Advances in neural information processing systems*, 20:553–560, 2007.

[7] 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.

[8] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[9] Tamir Hazan and Tommi Jaakkola. On the partition function and random maximum a-posteriori perturbations. *arXiv preprint arXiv:1206.6410*, 2012.

[10] Tamir Hazan, Francesco Orabona, Anand D Sarwate, Subhransu Maji, and Tommi S Jaakkola. High dimensional inference with random maximum a-posteriori perturbations. *IEEE Transactions on Information Theory*, 65(10):6539–6560, 2019.

[11] Uri Heinemann and Amir Globerson. What cannot be learned with bethe approximations. *arXiv preprint arXiv:1202.3731*, 2012.

[12] Aapo Hyvärinen. Consistency of pseudolikelihood estimation of fully visible boltzmann machines. *Neural Computation*, 18(10):2283–2292, 2006.

[13] Aapo Hyvärinen. Some extensions of score matching. *Computational statistics & data analysis*, 51(5):2499–2512, 2007.

[14] Vladimir Jojic, Stephen Gould, and Daphne Koller. Accelerated dual decomposition for map inference. In *ICML*, 2010.

[15] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. Mrf energy minimization and beyond via dual decomposition. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):531–552, 2010.

[16] Volodymyr Kuleshov and Stefano Ermon. Neural variational inference and learning in undirected graphical models. In *Advances in Neural Information Processing Systems*, pages 6734–6743, 2017.

[17] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database, 2010.

[18] Chongxuan Li, Chao Du, Kun Xu, Max Welling, Jun Zhu, and Bo Zhang. To relieve your headache of training an mrf, take advil. *arXiv preprint arXiv:1901.08400*, 2019.

[19] André FT Martins, Mário AT Figueiredo, Pedro MQ Aguiar, Noah A Smith, and Eric P Xing. Ad3: Alternating directions dual decomposition for map inference in graphical models. *The Journal of Machine Learning Research*, 16(1):495–545, 2015.

[20] Ofer Meshi, Mehrdad Mahdavi, and Alexander G Schwing. Smooth and strong: Map inference with linear convergence. In *NIPS*, pages 298–306. Citeseer, 2015.

[21] George Papandreou and Alan L Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision*, pages 193–200. IEEE, 2011.

[22] Zachary Pitkow, Yashar Ahmadian, and Ken Miller. Learning unbelievable probabilities. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[23] Carsten Rother, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. Optimizing binary mrfs via extended roof duality. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[24] Tatiana Shpakova. *On Parameter Learning for Perturb-and-MAP Models*. PhD thesis, PSL Research University, 2019.

[25] Tatiana Shpakova, Francis Bach, and Anton Osokin. Marginal weighted maximum log-likelihood for efficient learning of perturb-and-map models. *arXiv preprint arXiv:1811.08725*, 2018.

[26] Tomáš Šingliar and Miloš Hauskrecht. Noisy-or component analysis and its application to link analysis. *The Journal of Machine Learning Research*, 7:2189–2213, 2006.

[27] Charles Sutton and Andrew McCallum. Piecewise training for undirected models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 568–575. AUAI Press, 2005.

[28] Kevin Swersky, Daniel Tarlow, Ilya Sutskever, Ruslan Salakhutdinov, Richard Zemel, and Ryan Prescott Adams. Cardinality restricted boltzmann machines. *Advances in neural information processing systems*, 2012.

[29] Daniel Tarlow, Inmar Givoni, and Richard Zemel. Hop-map: Efficient message passing with high order potentials. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 812–819. JMLR Workshop and Conference Proceedings, 2010.

[30] Jakub M Tomczak, Szymon Zaręba, Siamak Ravanbakhsh, and Russell Greiner. Low-dimensional perturb-and-map approach for learning restricted boltzmann machines. *Neural Processing Letters*, 50(2):1401–1419, 2019.

[31] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching. In *AISTATS*, volume 3, page 3, 2003.

[32] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Map estimation via agreement on trees: message-passing and linear programming. *IEEE transactions on information theory*, 51(11):3697–3717, 2005.

[33] Huayan Wang and Koller Daphne. Subproblem-tree calibration: A unified approach to max-product message passing. In *International Conference on Machine Learning*, pages 190–198. PMLR, 2013.

[34] Yair Weiss, Chen Yanover, and Talya Meltzer. Map estimation, linear programming and belief propagation with convex free energies. *arXiv preprint arXiv:1206.5286*, 2012.

[35] Max Welling and Charles A Sutton. Learning in Markov random fields with contrastive free energies. In *AISTATS*, 2005.

[36] Sam Wiseman and Yoon Kim. Amortized bethe free energy minimization for learning mrfs. In *Advances in Neural Information Processing Systems*, pages 15546–15557, 2019.

[37] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Bethe free energy, kikuchi approximations, and belief propagation algorithms. *Advances in neural information processing systems*, 13, 2001.

[38] Jonathan S Yedidia, William T Freeman, Yair Weiss, et al. Generalized belief propagation. In *NIPS*, volume 13, pages 689–695, 2000.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 3, last paragraph.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] As shadowed areas in the graphs.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]