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

Miguel Lázaro-Gredilla, Antoine Dedieu, Dileep George
Vicarious AI
SF Bay Area, CA
{miguel, antoine, dileep}@vicarious.com

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. Very few works venture outside tractable models, and when they do, they use linear programming approaches, which as we 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 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 [37, 29, 33, 2, 13, 14, 12, 37, 17, 38, 19]. These methods typically fall into 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 [13]); (b) they can only be applied to fully observed models (e.g., piecewise training [29]); (c) they fail catastrophically for simple models (e.g., Bethe approximation [12]); (d) they require additional designs (such as a custom encoder) for new models, i.e., they are not black-box (e.g, Advil [19]).

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 [5]. Its main limitation is that for some models, mixing can be too slow to be usable. Recently the Gibbs-with-gradients (GWG) [8] sampler was proposed, which significantly improves mixing speed, enabling tractability of more models. A completely different approach to learning is Perturb-and-MAP [22], 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.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

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 $\{\phi_i(x_i;\theta_i) = \theta_i^\top \phi_i(x_i)\}_{i=1}^I$: 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. The energy of the model can be expressed as $E(x) = -\sum_{a=1}^A \phi_a(x_a;\theta_a) - \sum_{i=1}^I \phi_i(x_i;\theta_i)$ 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 $x \in \{0,\dots,M-1\}^D$, 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 [22], a family of samplers, including the following convenient form, was proposed:

$$x^{(s)} = \operatorname*{argmax}_x \left\{ \Theta_a^\top \Phi_a(x) + (\Theta_i + \varepsilon^{(s)})^\top \Phi_i(x) \right\} \quad (2)$$

where $\varepsilon^{(s)}$ is a random perturbation vector with as many entries as Θ_i (i.e., $\sum_i |x_i|$ entries, where $|x_i|$ is the cardinality of variable x_i). This means that to generate a sample, first we draw $\varepsilon^{(s)}$ from some distribution, use it to additively 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. We exclusively use independent, zero-mean densities $\varepsilon \sim \mathrm{Gumbel}(\varepsilon; -c, 1)$, where $c \approx 0.5772$ is the Euler-Mascheroni constant. With this choice, models with only unary terms result in the exact Gibbs distribution $p(x^{(s)}) \propto \exp(\Theta^\top \Phi(x^{(s)}))$ (see Lemma 1, [22]). More realistic models containing non-unary terms, will in general *not* match the Gibbs density, but will approximate it.

The Perturb-and-MAP approach also allows for the computation of an upper bound on the partition function, $\log \sum_x \exp(\Theta^\top \Phi(x)) \leq \left\langle \max_x (\Theta_a^\top \Phi_a(x) + (\Theta_i + \varepsilon)^\top \Phi_i(x)) \right\rangle_{p(\varepsilon)}$, (see Corollary 1,

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

$$\hat{\mathcal{L}}(\Theta|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}\left\langle \max_{x_h}(\Theta_a^\top \Phi_a(x^{(n)}, x_h) + (\Theta_i + \varepsilon)^\top \Phi_i(x^{(n)}, x_h))\right\rangle_{p(\varepsilon)} - \left\langle \max_{x}(\Theta_a^\top \Phi_a(x) + (\Theta_i + \varepsilon)^\top \Phi_i(x))\right\rangle_{p(\varepsilon)}. \tag{3}$$

whose gradient is $\nabla_\Theta \hat{\mathcal{L}}(\Theta|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N} \Phi(x^{(n)}, x_h)_{\hat{p}(x_h|x^{(n)};\Theta)} - \langle \Phi(x)\rangle_{\hat{p}(x;\Theta)}$, where $\hat{p}(\cdot;\Theta)$ is a (possibly conditional) sampler of the form of Eq. (2). 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 show in Section 5.2, even simple Ising models violate these assumptions.

2.3 The Bethe approximation and belief propagation

To make perturb-and-MAP more widely useful, we need a more general (even if approximate) approach to MAP inference. Here we describe a way to achieve this leveraging the Bethe free energy, which uses a variational formulation and a modified entropy to approximate the partition function.

Using an arbitrary variational distribution $q(x)$ we can lower bound the partition function using the expectation of the free energy. At the maximum with respect to this $q(x)$, we obtain the equality:

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

where $\mathcal{H}(q(x)) = -\sum_x q(x)\log q(x)$ is the entropy and β is an *inverse temperature* parameter. At $\beta = 1$, the l.h.s. of Eq. (4) becomes the log-partition function. At $\beta \to \infty$ and with perturbing Θ, the problem posed by Eq. (4) matches the one in Eq. (2). We keep this temperature parameter throughout our derivation. The maximization in (4) is still intractable, so we introduce two approximations and we obtain

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

which is the Bethe approximation of the log-partition function [39]. These approximations are (a) the use of the Bethe free entropy $\mathcal{H}_B(q_a(x_a), q_i(x_i)) = \sum_a \mathcal{H}(q_a(x_a)) + \sum_i (1 - f_i)\mathcal{H}(q_i(x_i))$ (which is exact for trees, approximate for loopy models) instead of the exact entropy; and (b) the use of locally consistent pseudo-marginals $q_a(x_a)$ and $q_i(x_i)$, which might not correspond to any joint density $q(x)$.

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 [40] 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 recover the primal quantities from the messages. In particular, the unary marginals are $q_i(x_i) \propto \exp(\beta(\phi_i(x_i;\theta_i) + \sum_{b\in\mathrm{nb}(i)} m_{b\to i}(x_i))$. The recovered primal value in Eq. (5), for $\beta = 1$, is an approximation of the log-partition function, but cannot be used for learning in general, see [23, 12], 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" $x_i = \arg\max_c \left\{\phi_i(x_i = c;\theta_i) + \sum_{b\in\text{nb}(i)} m_{b\to i}(x_i = c)\right\}$ 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) [7, 21, 16, 35, 20, 15, 34]. 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 [36]. 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 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 [36], and become the fixed points of max-product, so the multimodality of the problem is preserved, and hopefully one of the modes is 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 assess this in our experiments.

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 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 we 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: updates for messages *from* a factor or variable only involve messages *to* that factor or variable. In Algorithm 1, we use $\Theta|_{x^{(n)}}$ to refer to the modification[1] 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) [8]. GWG essentially optimizes the

[1] Observe that if for visible variables x_i we modify the unary parameters θ_i so as to attribute 0 score to the visible choice $x_i = x_i^{(n)}$ and $-\infty$ score to any other choice, the problem of finding the MAP for that new $\Theta|_{x^{(n)}}$ is effectively the same as finding the MAP for the original Θ, but clamping the visible variables to $x^{(n)}$.

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 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 finds 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 (the max-product updates for the factor are tractable). Fortunately, most small factors and some not-so-small ones[2] are, and they can be combined without limitation into rich EBMs. Neural functions are typically not easy to max-marginalize, and therefore using a whole neural network (NN) to define the energy is not compatible with PMP. A NN that is broken into max-marginalizable pieces, however, can be used.

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 uses 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

$\Theta \sim \mathcal{N}(0, 0.01)$ `// Init`
for $l \leftarrow 1$ ***to*** L **do**
  for $s \leftarrow 1$ ***to*** S **do**
    $n \leftarrow \text{Uniform}(1, N)$ `// Choose a sample from the training set`
    $y_{+}^{(s)} \leftarrow \texttt{pmp}\left(\Theta|_{x^{(n)}}, T\right)$ `// Get a sample from the posterior`
    $y_{-}^{(s)} \leftarrow \texttt{pmp}\left(\Theta, T\right)$ `// Get a sample from the prior`
  end
  $\Theta \leftarrow \Theta + \eta \left(\frac{1}{S} \sum_{s=1}^{S} \Phi(y_{+}^{(s)}) - \frac{1}{S} \sum_{s=1}^{S} \Phi(y_{-}^{(s)}) \right)$ `// Stochastic gradient ascent`
end
return Θ

Procedure `pmp(`Θ, T`)`
  $\varepsilon_i \sim \text{Gumbel}(-c, 1)\ \ \forall i;\ \ m_{i \to a}^{(0)}(x_i) \leftarrow 0\ \ \forall a, i, x_i;\ \ m_{a \to i}^{(0)}(x_i) \leftarrow 0\ \ \forall a, i, x_i$ `// Init`
  for $t \leftarrow 1$ ***to*** T **do**
    $m_{i \to a}^{(t)}(x_i) \leftarrow \phi_i(x_i; \theta_i + \varepsilon_i) + \sum_{b \in \texttt{nb}(i) \setminus a} m_{b \to i}^{(t-1)}(x_i)\ \ \forall a, i, x_i$ `// Max-product`
    $m_{a \to i}^{(t)}(x_i) \leftarrow \frac{1}{2} m_{a \to i}^{(t-1)}(x_i) + \frac{1}{2} \max_{x_{k \setminus i}} \left(\phi_a(x_a; \theta_a) + \sum_{k \in \texttt{nb}(a) \setminus i} m_{k \to a}^{(t)}(x_k) \right)\ \ \forall a, i, x_i$
  end
  $x_i \leftarrow \arg\max_c \left\{ \phi_i(x_i = c; \theta_i + \varepsilon_i) + \sum_{b \in \texttt{nb}(i)} m_{b \to i}(x_i = c) \right\}\ \ \forall i$ `// Decode`
  return x

4 Related work

Learning using Perturb-and-MAP has been proposed in different variants in recent literature, [22, 10, 6, 26, 25, 11, 25], 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 argue in Section 2.4.1, this approach does not work on multimodal problems (such as those invariant to permutations like in Section 5.6) and performs poorly even in simple settings, as we show in Sections 5.2 and 5.3.

In the seminal work [22], the authors suggest using QPBO [24] for the MAP problem, which, as a convex method, also fails even in small Ising models. The approaches in [10, 11] use the max-product LP (MPLP) [7] to solve the MAP problem, whose poor performance in non-attractive problems we show in Section 5.2. The recent work [25] is limited to tractable models. The structured learning of [26] also relies on log-supermodular potentials that make MAP inference tractable.

[2]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 [20], cardinality potentials limiting k out of n variables to be ON need $\mathcal{O}(kn)$ [30], other higher-order potentials accept efficient computation [31], etc.

A different approach is proposed in [6]: 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). The parameters of the perturbation are then 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. This approach has the advantage of being able to handle any noise distribution, but it is very computationally demanding and does not support Gumbel-distributed perturbations.

5 Experiments

We now study the empirical performance of PMP for sampling and learning. We measure performance using the maximum mean discrepancy (MMD) [9] between a set of model samples x and a set of ground truth samples x':

$$\text{MMD}^2(x, x') = \frac{1}{NN'} \sum_{i=1}^{N} \sum_{j=1}^{N'} k(x^{(i)}, x'^{(j)}) + k(x^{(i)}, x'^{(j)}) - 2k(x^{(i)}, x'^{(j)})$$

where $k(x^{(i)}, x'^{(j)}) = \exp - \left(\frac{1}{D} \sum_{d=1}^{D} [x_d^{(i)} \neq x'^{(j)}_d] \right)$ is the standard average Hamming distance kernel. MMD is also used in the original GWG paper [8]. Annealed importance sampling, while popular, is inadequate for this use case, see Appendix B.4 for details.

We 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.

Our experiments are run on a GeForce RTX 2080 Ti. All compared methods are coded on JAX [4] and run with changes only in the samplers for maximum comparability of their runtimes. Code to reproduce these experiments can be found at `https://github.com/vicariousinc/perturb_and_max_product/`. The specific max-product updates used in the presented models are derived in Appendix C. Code automating max-product inference in general models can be found in [41].

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 [23].

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 $\theta > 0.316$ [23]. 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 use Adam for 200 iterations with 0.01 learning rate: each iteration considers 100 chains and run 100 full sweeps. The result is $\hat{\theta}_{\text{PMP}} \approx 0.331$. One can 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 (a) the Gibbs distribution $\text{KL}(p(x|\theta = 0.5) \;||\; p(x|\hat{\theta}_{\text{PMP}} = 0.331)) = 0.119$, is much worse than to (b) the learned sampling distribution $\text{KL}(p(x|\theta = 0.5) \;||\; p_{\text{PMP}}(x|\hat{\theta}_{\text{PMP}} = 0.331)) \approx 0.008$, 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 how 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" [7], Globerson et al. show how the LP in Eq. (5) (for $\beta \to \infty$) 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. We

first consider a 2D cyclical square lattice with energy $E(x) = -0.1 \sum_{i,j \in \text{edge}} x_i x_j$, and side sizes $\{5, 10, 15\}$. We then generate 100 fully-connected Ising model with dimensions $\{5, 10, 15\}$ and 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 $x \in \{-1, +1\}$. For both problems, we calculate the exact partition function and estimations using PMP and MPLP (see Section 3). We run PMP for 200 full sweeps, and MPLP until the message updates change by less than 10^{-6} (which corresponds to more than 200 full sweeps in practice). The same perturbations are used for both methods, so they both solve the same exact MAP problem. The estimations returned are upper bounds, so the approximation error should be positive. In Figure 1[left] we see that this is the case for the 2D lattice, where both approaches perform similarly. However, for Ising models, neither method provides an actual upper bound (since the MAP inference and the expectation are only computed approximately), but MPLP is significantly worse, see Figure 1[center]. Furthermore, MPLP is a sequential algorithm (for which the damping trick used to turn max-product into a parallel algorithm does not work in practice): for the 15×15 2D lattice sampling from MPLP uses $500\times$ more compute time than PMP, making it impractical for sampling.

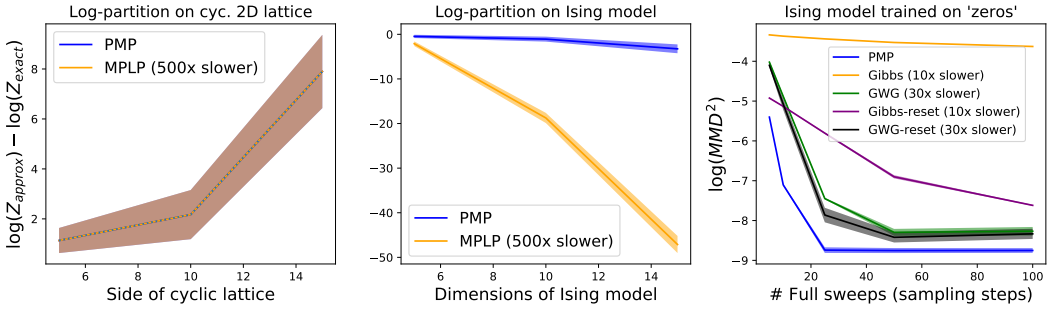


Figure 1: [Left and center] Log-partition estimation error using PMP and MPLP. Closer to zero is better. Shaded area represents ± 1 std deviation. [Right] See section 5.3. Sampling quality for different methods, keeping training fixed and varying the number of full sweeps at sampling time.

5.3 Ising model: MNIST zeros

In this experiment, we train a fully connected Ising model on a dataset of contours derived from the 0s of the MNIST dataset [18] (see top row of Figure 4[bottom left]). For learning, we initialize all the parameters to 0 and use Adam for 1000 iterations with 0.001 learning rate. At every iteration, we consider 100 chains and run 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 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.

We also use an LP solver (instead of max-product) with Pertub-and-MAP to learn the Ising model (see Appendix D). This approach reaches a $\log(\text{MMD}^2)$ of $-4.33(\pm 0.20)$. LP performance is poor and the approach is unpractical: it took three orders of magnitude more compute. See the LP samples at the bottom row of Figure 4[bottom, left] and how they compare with the other methods.

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

Here we reproduce the experiments from [8] with structured models, following precisely the definition, sampling, and learning described. See details in Appendix B. The authors used the root-mean-square error (RMSE) of the estimated parameters (which we include for reference) as their performance

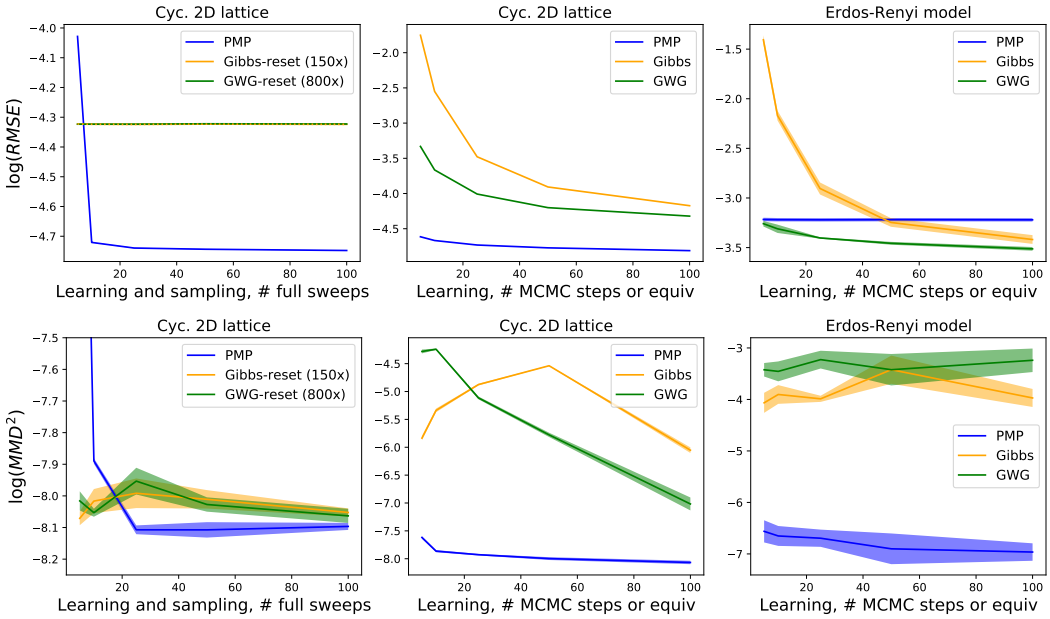


Figure 2: Performance on the lattice model ($\theta = -0.1$, side 25) and the Erdös-Renyi graphs from [8]. [Left column] x-axis determines the number of full sweeps for training and sampling, for all methods. [Center and right columns] x-axis determines the number of *individual variable updates* for 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.

metric, but this 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 [8] 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 [8], 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). 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 the highest sampling quality for any of the methods tested. Hence, despite being faster and more commonly used, this regime might not be the best choice for all applications.

5.5 Restricted Boltzmann machines: learning and sampling

Here we train a restricted Boltzmann machine (RBM) on MNIST data and sample from it using PMP. Its energy is defined as $E(x_v, x_h) = -\sum_{i,j} w_{ij} x_{vi} x_{hj} - \sum_j b_j x_{hj} - \sum_i c_i x_{vi}$, with $x_v \in \{0,1\}^{N_v}, x_h \in \{0,1\}^{N_h}$. To the best of our knowledge, PMP is the first reported instance of a perturb-and-MAP method able to train an RBM and sample recognizable MNIST-like digits from it. Indeed, we ran the author's code from [32] and the samples did not look like MNIST digits by a far margin. Note that no samples are reported in that work. See Appendix B.4 for discussion.

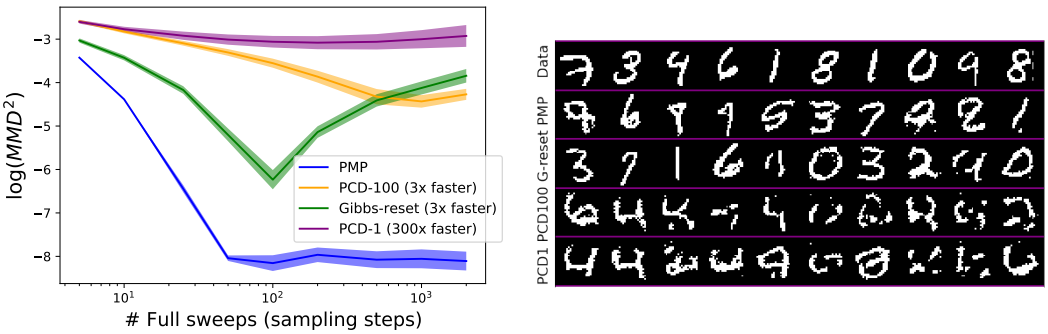


Figure 3: [Left] Sampling quality for an RBM trained once and sampled for an increasing number of full sweeps. [Right] Samples per method at 100 full sweeps. See Appendix B.2 for more samples.

We train an 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 initialize $W \sim \mathcal{N}(0, 0.01)$ and $b, c \sim \mathcal{N}(0, 1)$. 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, PMP is not. It is processing all factors in parallel, unaware of the conditional independence 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 B.2. Similar to Section 5.3, we tried to use an LP solver instead of max-product. We were able to get it to work to some extent on a smaller dataset, as reported in Appendix B.3, but the LP approach was still outperformed by PMP.

5.6 2D blind deconvolution

Finally, we consider the problem of sampling from an EBM with 40680 binary variables that are intricately connected, and which are invariant to permutations.

A binary 2D convolution generative model combines two sets of binary variables W, and S to form a set of binary images X. We illustrate in the top row of Figure 4 for the generation of 2 independent images. Each binary entry of W and S is modeled with an independent Bernoulli prior. The fact that the pixels in W are contiguous and form lines is not captured by the model, and is done only to produce shapes that humans can parse more easily.

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. The example in the top row of Figure 4 shows how the features W on the left are placed at the locations marked by S, with the top row of S producing the top image in X and the bottom row of S producing the bottom image in X. Each column of S corresponds to the locations of each of the 5 available features in W.

Observe that, unlike S, W is shared by all images. This makes all variables highly interdependent and the posterior highly multimodal: 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. The

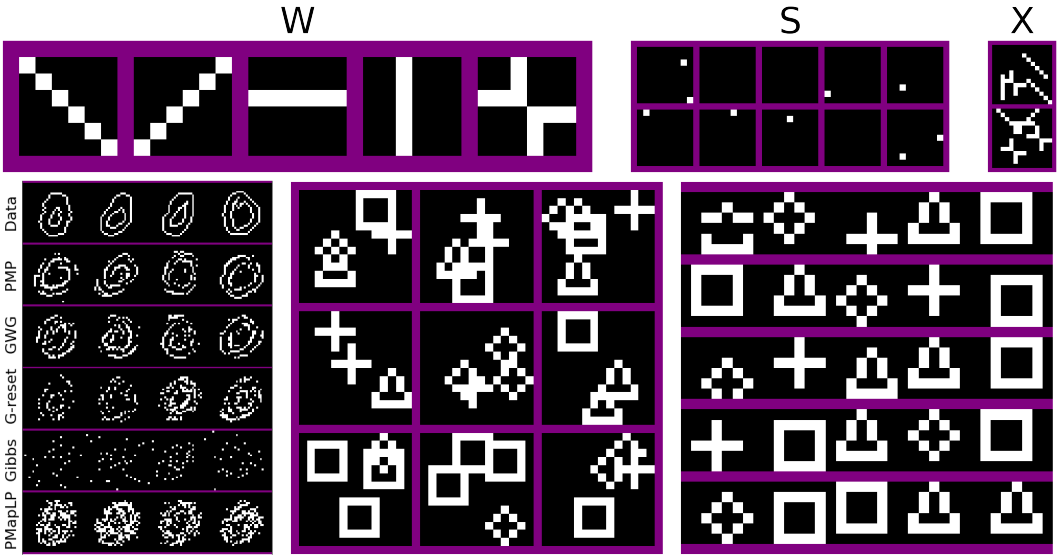


Figure 4: Top row: Binary 2D convolution example, with features (W), locations (S) and resulting images (X). Bottom row: [Left] Data and samples from MNIST zeros Ising model after 25 full sweeps of sampling using each tested method. [Center] 9 examples of blind deconvolution dataset (full dataset X in the Appendix B). [Right] 5 samples of the posterior using PMP.

proposed problem is a Bayesian version[3] of the task in [27], where the weights have been converted into binary variables. Note that Boolean matrix factorization emerges as a particular case.

Since this model has no free parameters, we do not need to train it. Instead, we directly turn to inference: given X (which contains 100 images generated according to a binary 2D convolution), we sample from the joint posterior $p(W, S|X)$. Out of those 100 images, 9 are shown in Figure 4[bottom row, center]. On its right, 5 samples (one per row) from the posterior are 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 dimensions of W used for the generation of X were $4 \times 5 \times 5$, but during inference they were set to $5 \times 6 \times 6$ to simulate a more realistic scenario in which we do not know their ground truth values. The samples from the posterior used that additional feature to include an extra copy of the symbols used for generation, but otherwise looked correct. The size of S was set to $100 \times 5 \times 9 \times 9$, so that the convolution output matches the size of X. 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 S or W were clamped to the true values, but not when trying to sample from both simultaneously. All 100 images in X are provided in Appendix B.

6 Discussion

In this work, we present PMP, a method for sampling and learning in EBMs. PMP improves over standard, LP-based Perturb-and-MAP in that it offers a MAP solver that is fast, parallel, anytime, and produces high quality solutions in problems where LP-based MAP solvers fail (e.g., simple non-attractive Ising models and challenging models with symmetries, such as the one in Section 5.6).

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. PMP also has the ability to solve some problems exponentially faster than Gibbs sampling, both 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.

[3] A different way to pose this problem is to assume that W is a set of parameters of the generative model, and infer them via maximum likelihood. Because we place a prior on them and infer their posterior instead, we are doing full Bayesian inference.

Acknowledgments and Disclosure of Funding

We thank Wolfgang Lehrach and Guangyao Zhou for useful discussions during the preparation of this manuscript.

This work was supported by Vicarious AI and ONR grant N00014-19-1-2368.

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] Christopher M Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

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

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] David Alexander Sontag. *Approximate inference in graphical models using LP relaxations*. PhD thesis, Massachusetts Institute of Technology, 2010.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

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

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

[39] 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.

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

[41] Guangyao Zhou*, Nishanth Kumar*, Miguel Lázaro-Gredilla, and Dileep George. PGMax: Factor graph on discrete variables and hardware-accelerated differentiable loopy belief propagation in JAX. `http://github.com/vicariousinc/PGMax`, 2021.

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]

Appendices

A Persistent PMP

During learning, persistent Gibbs sampling chains allow obtaining samples that are close to the target density in very few steps (typically only one) by warm-starting the sampling from the samples at the previous learning iteration. The idea is that since the weight update is small, the target distribution cannot have changed too much, and thus very few sampling steps are needed to take the Gibbs chain back to convergence at the new equilibrium distribution. This is the basis of persistent contrastive divergence, which significantly speeds up training when compared with bringing Gibbs sampling to convergence from scratch at each learning iteration.

PMP as presented in this work is not persistent, so it does not enjoy this benefit, although it is still very fast (but note that the lack of persistency might result in higher quality samples, as shown in Section 5). Here we want to show how we can make PMP compatible with persistency during learning. In principle, at each learning iteration we use a brand new perturbation, which means that the MAP problem is completely new and the solutions from previous steps cannot be leveraged to accelerate the current iteration. However, rather than creating an independent perturbation at each learning iteration, we could "perturb the previous perturbation" to obtain a very similar one. If both the perturbation and the weights are very similar to the previous ones, then the MAP problem as a whole is also very similar to the previous one, and we can bootstrap it from the solution obtained in the previous learning iteration.

As to the question of how to generate correlated variables with marginal Gumbel densities, a simple solution is to generate a chain of correlated normal variables following the marginal density $\mathcal{N}(0,1)$ and then transform them to follow the Gumbel density:

$$
\begin{aligned}
\delta^{(t)} &\sim \mathcal{N}(0,1) \;\; \forall t \\
\gamma^{(0)} &\sim \mathcal{N}(0,1) \\
\gamma^{(t)} &= \sqrt{\rho}\gamma^{(t-1)} + \sqrt{1-\rho}\delta^{(t-1)} \\
\varepsilon^{(t)} &= \mathrm{Gumbel}^{-1}_{\mathrm{CDF}}(\mathrm{Normal}_{\mathrm{CDF}}(\gamma^{(t)})) \;\; \forall t.
\end{aligned}
$$

Here, all the $\gamma^{(t)}$ variables follow a Gaussian density of mean 0 and variance 1, with a degree of correlation controlled by $0 \leq \rho \leq 1$. For $\rho = 0$ they are independent, whereas for $\rho = 1$, they all take the same value. The same is true for $\varepsilon^{(t)}$, but they will be marginally distributed as Gumbel. Using a large enough ρ and a small enough η (the learning rate), the MAP problem barely changes between learning iterations and it can be warm-started from the solution of the previous iteration.

B Experimental details

B.1 Details of the structured models experiments in Section 5.4

The experiments of this section follow exactly the details from [8], some of which are taken from the paper and some of which are taken from the code.

About the Erdös-Renyi model:

- It consists of 200 binary variables randomly connected so that the average number of neighbors of a node is 4.
- The energy expression for this model is $E(x) = x^\top W x + b^\top x$, where W is symmetric and has zero diagonal. The binary variables x are in $\{-1,+1\}^{200}$. Note that the customary 0.5 factor on the pairwise weights is missing, so each weight is counted twice.
- When two nodes are connected, the weight of their connection is drawn from $\mathcal{N}(0, 0.25)$ (the weight will be counted twice, see the previous item).
- There is a constant bias term in all the nodes of 0.1 (this is known from the authors' code).

About the cyclic lattice model:

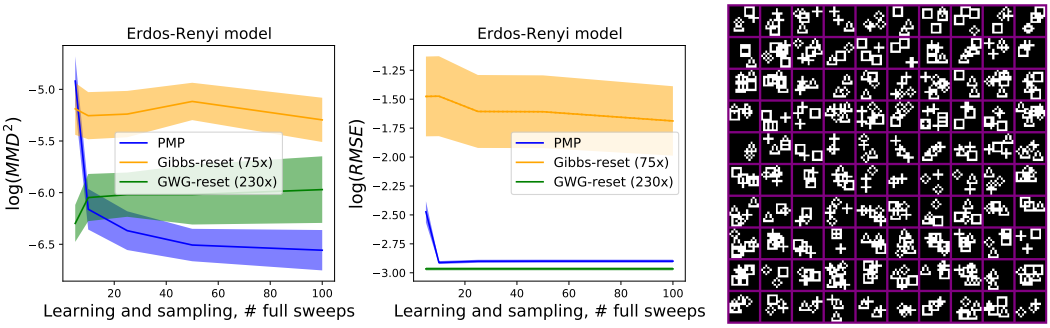


Figure 5: Left and center: Performance on the Erdös-Renyi graphs from [8] for non-persistent training. The x-axis determines the number of full sweeps for training and sampling, for all methods. Right: Full blind deconvolution dataset, 100 examples.

- It consists of 625 binary variables, connected in a periodic, cyclic 25×25 2D lattice. The average number of neighbors of a node is also 4.
- The energy expression for this model is $E(x) = x^\top W x$, where W is symmetric and has zero diagonal. The binary variables x are in $\{-1, +1\}^{625}$. Note that the customary 0.5 factor on the pairwise weights is missing, so each weight is counted twice.
- When two nodes are connected, the weight of their connection is set to θ (fixed to -0.1 in this work). However, that weight will be counted twice, see previous item.
- There is no bias term in this model.

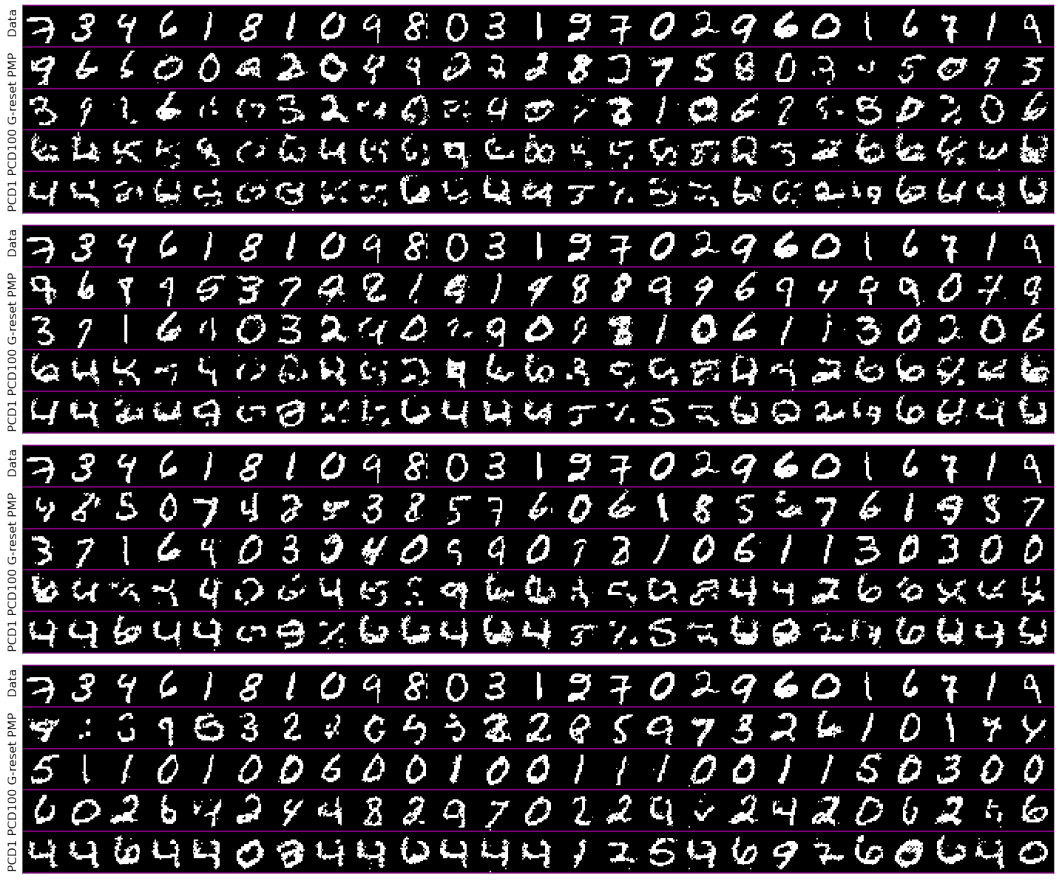


Figure 6: Data and 25 samples per method at {50, 100, 200, 2000} full sweeps (one per block).

For both models, the dataset is generated by running Gibbs sampling for 1,000,000 iterations, 2,000 times (thus generating 2,000 samples). The non-persistent training uses 1,000 learning iterations, with as many full sweeps as specified in between those. The persistent training follows [8] and uses 100,000 individual variable updates (or equivalent cost for PMP). We use Adam with learning rate 0.001 and regularize the weights with an $\ell 1$ penalty with strength 0.01. The structure of the model is not used for training, i.e., a completely general Ising model is learned. The root-mean-square error is computed between the ground truth matrix W and its estimation, considering all its entries (including the diagonal, which will be zero in both cases). The bias term is ignored.

We also provide the results with non-persistent training for the Erdös-Renyi graph in Figure 5[left and center], analogous to the ones provided in Figure 2[left] for the cyclic 2D lattice.

B.2 Additional samples from the RBM experiments in Section 5.5

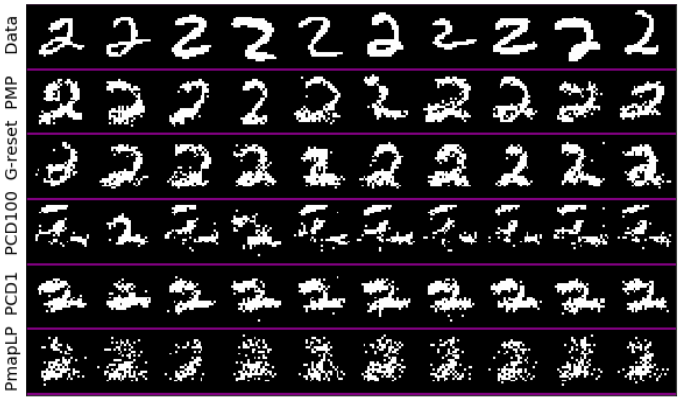
In the main paper, we provide 10 samples after 100 full sweeps for all the considered sampling methods. It is interesting to see the visual quality of the samples after different numbers of full sweeps, so we provide 25 samples at multiple numbers of full sweeps in Figure 6.

Note that visual quality can be misleading. For instance, when looking at the PCD1-trained model sampled for 2,000 full sweeps of Gibbs sampling (last block, last row), the subjective quality of each individual sampled digit might not be stellar, but it is acceptable. However, the performance as measured by MMD is dismal. This might seem counterintuitive until we realize the lack of diversity in the sampled digits, with a strong bias towards a particular version of digit 4. Even if the 4 is acceptable, its relative weight is way too high, thus explaining the poor density match measured by MMD.

B.3 RBM samples on a single digit: comparison with LP solver

We propose an additional experiment where we train an RBM with 250 hidden variables on the 2s on the MNIST dataset and compare PMP and other methods with the LP solver. Because the LP solver is prohibitively slow, we only train for 1000 iterations, using Adam with learning rate 0.01. We use an Amazon Web Service c5d.24xlarge instance with 96 CPUs for LP training. Every iteration consider 50 chains, and we use 100 sweeps for the different sampling methods. We report the MMDs and present samples from the different methods in Figure 7.

PMP still achieves the best MMD and produces a rich variety of samples, with different shapes as in the MNIST dataset. The LP solver is the second best method in terms of MMD. It is however three orders of magnitude slower than PMP, and does not capture the same variety of samples.



Method	MMD
PMP	**-5.36** (0.06)
Gibbs-reset	-4.18 (0.05)
PDC100	-2.70 (0.22)
PDC1	-2.88 (0.22)
PmapLP	-4.38 (0.27)

Figure 7: [Left] RBM samples for the different methods after training on 2s only for 1000 iterations. [Right] Corresponding MMD: we use 100 full sweeps for the sampling methods.

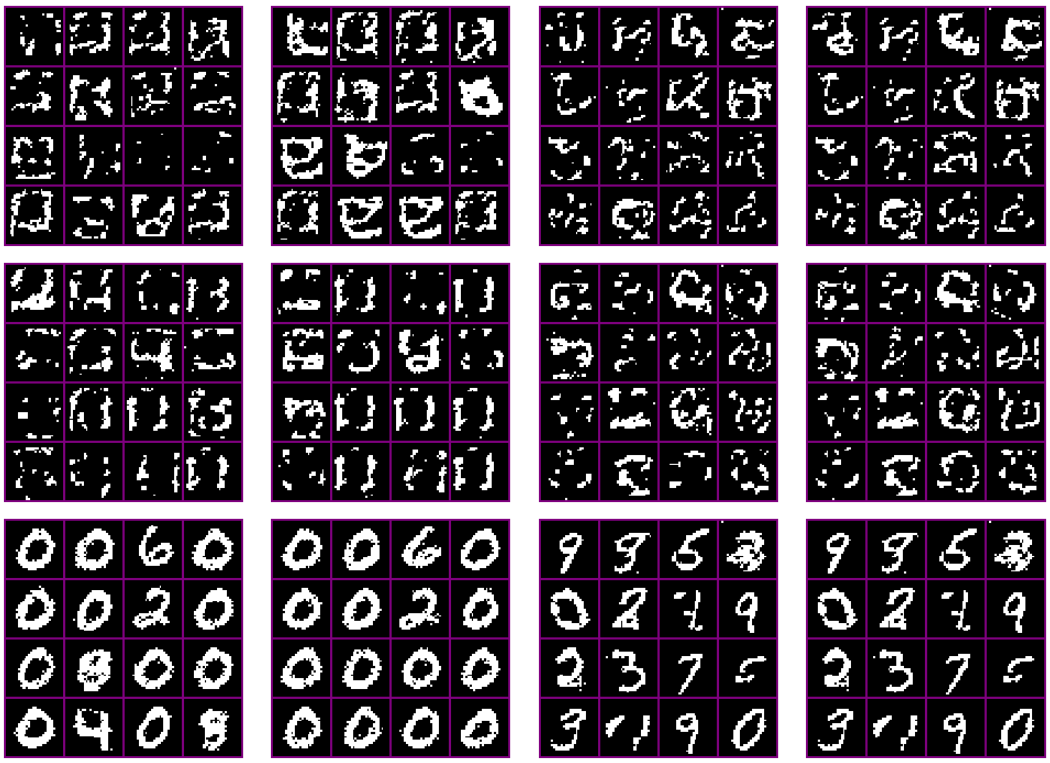
B.4 About previous work on perturb-and-MAP training of RBMs

[32] presents a Perturb-and-MAP-based method for RBM learning. It is also used to learn an RBM with 500 hidden units on MNIST digits. The authors report success by providing Annealed Importance Sampling (AIS) figures that are very similar to those obtained with contrastive divergence. They however do not show any samples from their trained RBMs.

Since the authors released their code, we ran it (with their chosen parameters) and tried to sample from the resulting RBM weights. We tried Gibbs sampling (which makes sense if AIS is a valid way to measure the quality of the model), PMP (which makes sense if the weights are to be interpreted as valid for perturb-and-MAP sampling) and coordinate descent (the method proposed by the authors). We tried 100 and 10,000 steps for each of these methods. The same approach is applied to the RBM we trained in Section 5.5. Results are displayed on Figure 8. For coordinate descent samples very few pixels turned on, so we did not include them.

We were not able to obtain any meaningful MNIST samples from the RBM trained according to [32]. A possible explanation is that their method actually did not properly train the RBM, and yet AIS was incorrectly reporting a good figure, since AIS is known to be a (sometimes very) optimistic approach to log-likelihood estimation. This optimism might also be due to inadequate hyperparameters within AIS. We were in fact able to obtain extremely (and incorrectly) good AIS scores when training with PMP. In light of this, we decided to use MMD in this work because it is neither optimistic nor its value depends on other hyperparameters that have to be carefully tuned (such as the ones that control the mixing of AIS).

Based on the above, we believe this to be the first work to successfully train an RBM on the MNIST dataset using perturb-and-MAP, and it is definitely the first one to do that and that can sample recognizable, MNIST-like digits using perturb-and-MAP.



(a) Gibbs, 100 it. (b) Gibbs, 10000 it. (c) PMP, 100 it. (d) PMP, 10,000 it.

Figure 8: [First row] Trained with perturb and coordinate descent, 1 step. [Second row] Trained with perturb and coordinate descent, 10 step. [Third row] Trained with PMP, 100 full sweeps.

C Max-product updates for Ising models, RBM, OR and AND factors

We derive efficient and numerically robust max-product update equations for Ising and RBM models, as well as for the OR and AND factors used in the Section 5.6.

C.1 Ising models

We consider a binary Ising model over n binary variables with energy:

$$E(x) = -\frac{1}{2}x^T W x - b^T x, \;\; \forall x \in \{0,1\}^n. \tag{5}$$

This energy defines a probability $p(x) = \frac{1}{Z}e^{-E(x)}$, where Z is the partition function. By factorizing this probability over the nodes and edges of the Ising graph, we observe that two different potentials are involved: (a) joint potentials $\phi_{ij}(x_i, x_j) = \exp(w_{ij}x_i x_j) \;\; \forall i, j$ and (b) unaries potentials $\theta_i(x_i) = \exp(b_i x_i) \;\; \forall i$. It then holds:

$$p(x) = \frac{1}{Z}\prod_i \theta_i(x_i) \prod_{i \neq j} \phi_{ij}(x_i, x_j).$$

BP updates: The max-product update [3] for the message going from the ith to the jth variable (only defined for $j \neq i$) is defined as:

$$m_{i\to j}(x_j) \;\; \propto \max_{x_i \in \{0,1\}} \left\{ \theta_i(x_i)\phi_{ij}(x_i, x_j) \prod_{k \neq j} m_{k\to i}(x_i) \right\}, \;\; \forall x_j \in \{0,1\},$$

which is equivalent to saying that

$$m_{i\to j}(x_j) \;\; \propto \max \left\{ \theta_i(0)\phi_{ij}(0, x_j) \prod_{k \neq j} m_{k\to i}(0), \;\; \theta_i(1)\phi_{ij}(1, x_j) \prod_{k \neq j} m_{k\to i}(1) \right\}.$$

After running max-product for a fixed number of iterations, the beliefs are computed via:

$$b(x_i) \;\; \propto \theta_i(x_i) \prod_k m_{k\to i}(x_i),$$

and we assign each variable to the value with larger belief.

For numerical stability, we first normalize the messages such that:

$$\max(m_{i\to j}(0), \;\; m_{i\to j}(1)) = 1, \;\; \forall i, j.$$

Second, we map messages to the log-space and observe that, because of the above normalization, both $m_{i\to j}(0)$ and $m_{i\to j}(1)$ can be derived from a single message $n_{i\to j}$ defined as:

$$\begin{aligned}
n_{i\to j} &:= \log\left(m_{i\to j}(1)\right) - \log\left(m_{i\to j}(0)\right) \\
&= \max\left\{ \sum_{k\neq j} \log m_{k\to i}(0), \;\; b_i + w_{ij} + \sum_{k\neq j} \log m_{k\to i}(1) \right\} \\
&\quad - \max\left\{ \sum_{k\neq j} \log m_{k\to i}(0), \;\; b_i + \sum_{k\neq j} \log m_{k\to i}(1) \right\} \\
&= \max\left\{ 0, \;\; b_i + w_{ij} + \sum_{k\neq j} (\log m_{k\to i}(1) - \log m_{k\to i}(0)) \right\} \\
&\quad - \max\left\{ 0, \;\; b_i + \sum_{k\neq j} (\log m_{k\to i}(1) - \log m_{k\to i}(0)) \right\} \\
&= \max\left\{ 0, \;\; b_i + w_{ij} + \sum_{k\neq j} n_{k\to i} \right\} - \max\left\{ 0, \;\; b_i + \sum_{k\neq j} n_{k\to i} \right\}.
\end{aligned} \tag{6}$$

In addition, the MAP assignment is derived as follows:

$$x_i = 1 \text{ if } b_i + \sum_{k \neq i} n_{k \to i} \geq 0 \text{ and } x_i = 0 \text{ o.w.}$$

For computational efficiency we define $N \in \mathbb{R}^{n \times n}$ such that $N_{ij} = n_{i \to j}$ with $N_{ii} = 0$ and we observe that Eq. (6) can be expressed as:

$$\begin{aligned} P &= N^T 1_n 1_n^T - N^T + b 1_n^T \\ N &= \max(0_{n \times n},\ P + W) - \max(0_{n \times n},\ P). \end{aligned}$$

where $1_n \in \mathbb{R}^{n \times 1}$ and $0_{n \times n} \in \mathbb{R}^{n \times n}$ are a vector (resp. a matrix) of 1s (resp. 0).

C.2 Restricted Boltzmann machine

The energy of a binary RBM is $E(v, h) = -h^T W v - b^T h - c^T v$. The max-product updates in log-space can be derived similarly to the above. We now define n_{HV} (resp. n_{VH}) the messages going from the hidden variables to the visible ones (resp. from the visible to the hidden). We obtain:

$$n_{i \to j}^{HV} = \max \left\{ 0,\ \ b_i + w_{ij} + \sum_{k \neq j} n_{k \to i}^{VH} \right\} - \max \left\{ 0,\ \ b_i + \sum_{k \neq j} n_{k \to i}^{VH} \right\}$$

C.3 Max-product updates for `OR` and `AND` factors

We finally consider the `OR` and `AND` factors involved in Section 5.6. The bottom variable of an `OR` factor is 1 if and only if at least one variable is 1. The bottom variable of an `AND` factor is 1 if and only if all the variable are 1. In our experiments we only consider `AND` factor with two top-level variables.

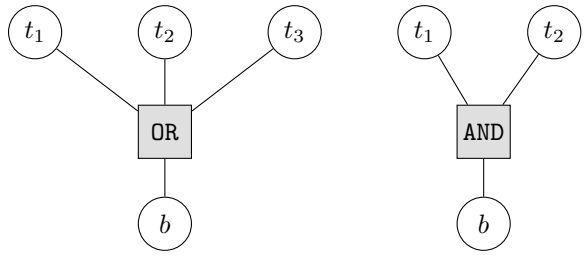


Figure 9: Factor graph for an `OR` factor [left] and an `AND` factor with two top-level variables [right].

Notations: We note $m_{f \to c}(x_c) > 0$ (resp. $m_{c \to f}(x_c) > 0$) the message in real space going from a factor f to a binary variable c with value $x_c \in \{0, 1\}$ (resp. from a variable c with value x_c to a factor f). We map messages to log space by defining $n_{f \to c} = \log(m_{f \to c}(1)) - \log(m_{f \to c}(0))$.

`AND` factor, forward pass: We derive herein the expression of the message going from the `AND` factor to the top variable t_1. By definition it holds

$$\begin{cases} m_{f \to t_1}(1) = \max\{m_{b \to f}(1)\ m_{t_2 \to f}(1),\ m_{b \to f}(0)\ m_{t_2 \to f}(0)\} \\ m_{f \to t_1}(0) = \max\{m_{b \to f}(0)\ m_{t_2 \to f}(0),\ m_{b \to f}(0)\ m_{t_2 \to f}(1)\} \end{cases}$$

It consequently holds:

$$\begin{aligned} n_{f \to t_1} &= \max\{\log m_{b \to f}(1) + \log m_{t_2 \to f}(1),\ \ \log m_{b \to f}(0) + \log m_{t_2 \to f}(0)\} \\ &\quad - \max\{\log m_{b \to f}(0) + \log m_{t_2 \to f}(0),\ \ \log m_{b \to f}(0) + \log m_{t_2 \to f}(1)\} \\ &= \max\{n_{b \to f} + n_{t_2 \to f}, 0\} - \max\{\ n_{t_2 \to f}, 0\} \end{aligned}$$

A similar expression hols for $n_{f \to t_2}$.

`AND` **factor, backward pass:** We consider herein the message going from the `AND` factor to the bottom variable b. By definition, we have:

$$\begin{cases} m_{f\to b}(1) = m_{t_1\to f}(1)\; m_{t_2\to f}(1) \\ m_{f\to b}(0) = \max\{m_{t_1\to f}(0)\; m_{t_2\to f}(0),\; m_{t_1\to f}(0)\; m_{t_2\to f}(1),\; m_{t_1\to f}(1)\; m_{t_2\to f}(0)\} \end{cases}$$

It consequently holds:

$$n_{f\to b} = \min\{n_{t_1\to f} + n_{t_2\to f},\; n_{t_1\to f},\; n_{t_2\to f}\}$$

`OR` **factor, forward pass:** We now consider the messages going from the `OR` factor to the top variables. To this end, we need to consider the variable with the largest incoming message separately. Let us assume that $m_{t_1\to f} = \max_i\{m_{t_i\to f}\}$ and that $m_{t_2\to f} = \max_{i>1}\{m_{t_i\to f}\}$. For $i > 1$ we have

$$\begin{cases} m_{f\to t_i}(1) = m_{b\to f}(1) \prod\limits_{j\neq i} \max\{m_{t_j\to f}(1),\; m_{t_j\to f}(0)\} \\ m_{f\to t_i}(0) = \max\left\{ m_{b\to f}(0) \prod\limits_{j\neq i} m_{t_j\to f}(0),\;\; m_{b\to f}(1)\, m_{t_1\to f}(1) \prod\limits_{j\neq 1,i} \max\{m_{t_j\to f}(1),\; m_{t_j\to f}(0)\} \right\} \end{cases}$$

It consequently holds:

$$\begin{aligned} n_{f\to t_i} &= n_{b\to f} + \sum_{j\neq i} \max\{0,\; n_{t_i\to f}\} - \max\left\{ 0,\; n_{b\to f} + n_{t_1\to f} + \sum_{j\neq 1,i} \max\{0,\; n_{t_i\to f} \right\} \\ &= \min\left\{ n_{b\to f} + \sum_{j\neq i} \max\{0,\; n_{t_i\to f}\},\;\; \max\{0,\; n_{t_1\to f}\} - n_{t_1\to f} \right\}. \end{aligned}$$

A similar reasoning for $i = 1$ gives:

$$n_{f\to t_1} = \min\left\{ n_{b\to f} + \sum_{j\neq 1} \max\{0,\; n_{t_i\to f}\},\;\; \max\{0,\; n_{t_2\to f}\} - n_{t_2\to f} \right\}.$$

`OR` **factor, backward pass:** We finally derive an expression for the message going from the `OR` factor to the bottom variable b. We also assume that $m_{t_1\to f} = \max_i\{m_{t_i\to f}\}$. It then holds:

$$\begin{cases} m_{f\to b}(1) = m_{t_1\to f}(1) \prod\limits_{j\neq 1} \max\{m_{t_j\to f}(1),\; m_{t_j\to f}(0)\} \\ m_{f\to b}(0) = \prod\limits_{j} m_{t_j\to f}(0), \end{cases}$$

from which we derive

$$n_{f\to b} = n_{t_1\to b} + \sum_{j\neq 1} \max\{0,\; n_{t_i\to f}\}.$$

D LP relaxations for MAP inference

In this section, we derive efficient LP relaxations for MAP inference in Ising and RBM models. The formulations we propose are equivalent to the standard LP relaxation and produce the same exact relaxed solutions for x, but does so faster.

D.1 LP relaxations for Ising models

For a fully connected binary Ising model with energy given by Eq. (5), the MAP problem is defined as:

$$\max_{x\in\{0,1\}^n} \sum_{1\leq i\neq j\leq n} \frac{1}{2} w_{ij} x_i x_j + \sum_{i=1}^{n} b_i x_i. \tag{7}$$

We denote by $\{q_{ij}(x_i, x_j) \mid 1 \leq i \neq j \leq n;\ x_i, x_j \in \{0, 1\}\}$ a distribution over the edges of the model and $\{p_i(x_i) \mid 1 \leq i \leq n;\ x_i \in \{0, 1\}\}$ a distribution over the nodes. The standard LP relaxation [28] of Problem (7) is:

$$\begin{aligned}
\max_{p,q} \quad & \sum_{1 \leq i \neq j \leq n} \sum_{0 \leq x_i, x_j \leq 1} \tfrac{1}{2} w_{ij} q_{ij}(x_i, x_j) + \sum_{i=1}^{n} \sum_{0 \leq x_i \leq 1} b_i p_i(x_i). \\
& \textstyle\sum_{0 \leq x_i \leq 1} p_i(x_i) = 1 && \forall i \\
& \textstyle\sum_{0 \leq x_i, x_j \leq 1} q_{ij}(x_i, x_j) = 1 && \forall i \neq j \\
& p_i(x_i) = \textstyle\sum_{0 \leq x_j \leq 1} q_{ij}(x_i, x_j) && \forall i \neq j \\
& p_j(x_j) = \textstyle\sum_{0 \leq x_i \leq 1} q_{ij}(x_i, x_j) && \forall i \neq j \\
& 0 \leq p \leq 1, \quad 0 \leq q \leq 1.
\end{aligned}$$

The above LP involves $2n + 4n(n-1) = 4n^2 - 2n$ variables and $9n^2 - 6n$ constrains. We propose to solve the smaller and equivalent LP with n^2 variables $\{\tilde{p}_i\}_{1 \leq i \leq n} \cup \{\tilde{q}_{ij}\}_{1 \leq i \neq j \leq n}$ and $4n^2 - 3n$ constrains defined as:

$$\begin{aligned}
\max_{\tilde{p}, \tilde{q}} \quad & \sum_{1 \leq i \neq j \leq n} w_{ij} \tilde{q}_{ij} + \sum_{i=1}^{n} b_i \tilde{p}_i. \\
& \tilde{q}_{ij} \leq \tilde{p}_i && \forall i \neq j \\
& \tilde{q}_{ij} \leq \tilde{p}_j && \forall i \neq j \\
& \tilde{p}_i + \tilde{p}_j - 1 \leq \tilde{q}_{ij} && \forall i \neq j \\
& \tilde{p} \leq 1,\ 0 \leq \tilde{q}.
\end{aligned}$$

Proof: To prove the equivalence, start from a feasible solution $(\tilde{p}, \tilde{q})$ of the second problem and define $p_i(1) = \tilde{p}_i,\ p_i(0) = 1 - \tilde{p}_i$ and $q_{ij}(1,1) = q_{ij},\ q_{ij}(1,0) = \tilde{p}_i - q_{ij},\ q_{ij}(0,1) = \tilde{p}_j - \tilde{q}_{ij},\ q_{ij}(0,0) = 1 - \tilde{p}_i - \tilde{p}_j + \tilde{q}_{ij}$ and observe that we obtain a feasible solution (p, q) of the first problem with same objective value.

D.2 LP relaxations for RBMs

We consider a binary RBM with m hidden and n visible variables. We introduce the variables $\{\tilde{h}_i\}_{1 \leq i \leq m} \cup \{\tilde{v}_j\}_{1 \leq j \leq n} \cup \{\tilde{z}_{ij}\}_{1 \leq i \leq n,\ 1 \leq j \leq m}$. Similar to above, the LP relaxation of the MAP assignment problem can be expressed as:

$$\begin{aligned}
\max_{h, v, z} \quad & \sum_{1 \leq i \neq j \leq n} w_{ij} \tilde{z}_{ij} + \sum_{i=1}^{n} b_i \tilde{h}_i + \sum_{j=1}^{n} c_j \tilde{v}_j \\
& \tilde{z}_{ij} \leq \tilde{h}_i && \forall i \neq j \\
& \tilde{z}_{ij} \leq \tilde{v}_j && \forall i \neq j \\
& \tilde{h}_i + \tilde{v}_j - 1 \leq \tilde{z}_{ij} && \forall i \neq j \\
& \tilde{h} \leq 1,\ \tilde{v} \leq 1,\ 0 \leq \tilde{z}.
\end{aligned}$$