Learning noisy-OR Bayesian Networks with Max-Product Belief Propagation

Antoine Dedieu [1] **Guangyao Zhou** [1] **Dileep George** [1] **Miguel Lázaro-Gredilla** [1]

Abstract

Noisy-OR Bayesian Networks (BNs) are a family of probabilistic graphical models which express rich statistical dependencies in binary data. Variational inference (VI) has been the main method proposed to learn noisy-OR BNs with complex latent structures (Jaakkola & Jordan, 1999; Ji et al., 2020; Buhai et al., 2020). However, the proposed VI approaches either (a) use a recognition network with standard amortized inference that cannot induce "explaining-away"; or (b) assume a simple mean-field (MF) posterior which is vulnerable to bad local optima. Existing MF VI methods also update the MF parameters sequentially which makes them inherently slow. In this paper, we propose parallel max-product as an alternative algorithm for learning noisy-OR BNs with complex latent structures and we derive a fast stochastic training scheme that scales to large datasets. We evaluate both approaches on several benchmarks where VI is the state-of-the-art and show that our method (a) achieves better test performance than Ji et al. (2020) for learning noisy-OR BNs with hierarchical latent structures on large sparse real datasets; (b) recovers a higher number of ground truth parameters than Buhai et al. (2020) from cluttered synthetic scenes; and (c) solves the 2D blind deconvolution problem from Lazaro-Gredilla et al. (2021) and variants—including binary matrix factorization—while VI catastrophically fails and is up to two orders of magnitude slower.

1. Introduction

Probabilistic graphical models (PGMs) propose a rigorous and elegant way to represent the full joint probability density function of high-dimensional data and to express assumptions about its hidden structure. Learning and inference algorithms let us analyze data under those assumptions and recover the hidden structure that best explains our observations. However, performing exact inference in complex PGMs is often intractable. To mitigate this problem, several techniques have been proposed for approximate inference, among which a popular one is variational inference (VI) (Wainwright et al., 2008; Bishop & Nasrabadi, 2006).

In this paper, we consider directed acyclic PGMs—also named Bayesian networks (BNs)—with binary variables and noisy-OR conditional distribution (Pearl, 1988). The resulting noisy-OR BNs have been used for medical diagnosis (Jaakkola & Jordan, 1999), data compression (Šingliar & Hauskrecht, 2006), text mining (Liu et al., 2016), and more recently overparametrized learning (Buhai et al., 2020) and topic modeling on large sparse datasets (Ji et al., 2020). Noisy-OR BNs have an intractable posterior: most of the aforementioned applications rely on VI for approximate inference. Some existing VI methods (Buhai et al., 2020) use a recognition network and amortize the approximate inference via a single forward pass, which cannot induce "explaining-away" (see Section 2). In contrast, Jaakkola & Jordan (1999); Šingliar & Hauskrecht (2006); Ji et al. (2020) assume a mean-field (MF) posterior, which is vulnerable to bad local optima. These existing MF methods also update the MF parameters sequentially—i.e. one by one—which is prohibitively slow. To scale MF VI, Ji et al. (2020) propose a local heuristic that updates fewer MF parameters (see Section 3). However, their approach only applies to large sparse datasets (i.e. most of the observations are 0s).

In this work, we propose a fast and efficient stochastic scheme for learning noisy-OR BNs that use the parallel max-product (MP) algorithm (Pearl, 1988; Murphy et al., 2013) as an alternative to VI. We scale MP to very large noisy-OR BNs by reducing the max-product updates complexity for a noisy-OR factor from exponential to linear in the number of variables. We accelerate MP on Graphics Processing Units (GPUs) using a recent open-sourced package (Zhou et al., 2022). Similar to Ji et al. (2020), our method supports multi-layered noisy-OR networks and relies on stochastic optimization (Robbins & Monro, 1951) for scaling. However, (a) contrary to Ji et al. (2020), our approach runs in parallel which allows it to scale to large dense datasets; and (b) in contrast with Buhai et al. (2020), our method induces

[1]DeepMind. Correspondence to: Antoine Dedieu <adedieu@deepmind.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

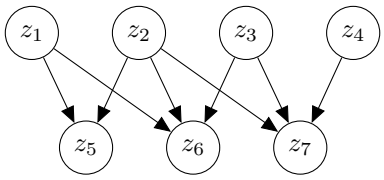

Figure 1. Two-layers noisy-OR Bayesian network with four hidden and three visible nodes. The leak node ($z_0 = 1$) is not shown.

explaining-away. For large sparse datasets where MF VI is the best existing method (Ji et al., 2020), we show that our approach efficiently explores the parameters space, and can be used as a powerful initialization for MF VI to reach new state-of-the-art performance. We additionally show that several challenging problems including (a) binary matrix factorization; (b) the noisy-OR BNs experiments from Buhai et al. (2020); and (c) the complex 2D blind deconvolution problem from Lazaro-Gredilla et al. (2021) can be expressed as learning problems in noisy-OR BNs, for which standalone MP outperforms VI while being up to two orders of magnitude faster. Our code is written in `JAX` (Bradbury et al., 2018) and is available at `https://github.com/deepmind/max_product_noisy_or`.

The rest of this paper is organized as follows. Section 2 reviews noisy-OR BNs while Section 3 discusses existing learning methods for these models. Section 4 introduces the max-product algorithm, which Section 5 integrates into our training scheme for BNs. Finally, Section 6 compares our method with VI in a wide variety of experiments.

2. Noisy-OR Bayesian networks

Given binary observations $x \in \{0, 1\}^p$, we model its statistical dependencies using binary BNs (Koller & Friedman, 2009) as in Figure 1. The nodes in the graph are divided into p visible nodes—which are the leaves—and m hidden nodes. Each visible (resp. hidden) node i is associated with a binary random variable x_i (resp. h_i). We denote $h = (h_1, \dots, h_m)$ and $x = (x_1, \dots, x_p)$. Similar to Ji et al. (2020), we introduce a leak node 0 that connects to all the nodes, whose variable z_0 is always active, i.e. $z_0 = 1$. The leak node allows any active variable to be explained by other factors than its parents. For convenience, we denote $z = (z_0, h, x)$ the vector of all variables: $h = (z_1, \dots, z_m)$ and $x = (z_{m+1}, \dots z_{m+p})$.

Let $\mathcal{P}(i)$ be the set of parents (excluding the leak node) of the node $i \geq 1$. The activation probability of the variable z_i is given by the noisy-OR conditional distribution

$$p(z_i = 0 \mid z_{\mathcal{P}(i)},\ \Theta) = \exp\left(-\theta_{0\to i} - \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\right) \tag{1}$$

where $\theta_{0\to i} \geq 0,\ \theta_{k\to i} \geq 0,\ \forall k \in \mathcal{P}(i)$ and Θ is the vector collecting all these parameters. This conditional distribution possesses three important properties. First, if all the parents are inactive, the activation probability is given by the leak node: $p(z_i = 0 \mid z_{\mathcal{P}(i)} = 0,\ \Theta) = \exp(-\theta_{0\to i})$. As in Buhai et al. (2020), we refer to $1 - \exp(-\theta_{0\to i})$ as the "prior probability" when $\mathcal{P}(i)$ is empty and the "noise probability" otherwise. Second, if only the variable k is connected to the variable i and there is no leak node, $p(z_i = 0 \mid z_k = 1,\ \Theta) = \exp(-\theta_{k\to i})$—which we refer to as the "failure probability". Finally, noisy-OR BNs can induce "explaining-away": explaining-away creates competition between a-priori unlikely causes, which allows inference to pick the smallest subset of causes that explain the effects.

3. Related Work

The QMR-DT network (Jaakkola & Jordan, 1999) is one of the first models which exploits the properties of noisy-OR BNs. It consists of a two-layer bipartite graph created by domain experts which models how 600 diseases explain $4,000$ findings. The probability of a finding given diseases is expressed by Equation (1). After learning, the QMR-DT network is used to infer the probabilities of different diseases given a set of observed symptoms. For approximate inference in the intractable noisy-OR BN, the authors assumed a MF posterior—which can induce explaining-away (see Section 2)—and introduced a family of variational bounds as well as a heuristic to increase the graph sparsity.

Other approaches have been proposed for learning bipartite noisy-OR BNs. Šingliar & Hauskrecht (2006) introduced a variational EM procedure that exploits the bounds of Jaakkola & Jordan (1999) while assuming a fully connected graph. Halpern & Sontag (2013) proposed a method of moments that requires the graph to be sparse. Liu et al. (2016) introduced a Monte-Carlo EM algorithm that requires a large number of sampling steps for good performance. None of these methods would scale to large datasets.

Recently, Buhai et al. (2020) discussed the effect of overparameterization in PGMs and showed that, on synthetic datasets, increasing the number of latent variables of noisy-OR BNs improves their performance at recovering the ground truth parameters. Their method considered VI with a recognition network. However, the authors amortize the inference via a single forward pass: inference results in picking all causes that are consistent with the effects and cannot induce explaining-away (see Section 2).

In another recent work, Ji et al. (2020) proposed a stochastic variational training algorithm for noisy-OR BNs. The authors assumed a MF posterior and extended the bounds of Jaakkola & Jordan (1999). For scalability, the authors introduced "local models": they only update the variational parameters associated with the ancestors of the active visible

variables. They showed that this is equivalent to optimizing a constrained variational bound, and derived state-of-the-art performance for multi-layered BNs on large sparse real datasets, while significantly outperforming Liu et al. (2016).

The method we propose in Section 5 for learning noisy-OR BNs has the same appealing properties as Ji et al. (2020): it induces explaining-away, it supports multi-layered graphs and it scales to large sparse datasets. In addition, (a) it is faster as it runs parallel max-product; (b) it also scales to large dense datasets; and (c) it considers a richer posterior than MF VI which allows it to find better local optima.

4. Background on Max-Product

We first review the parallel max-product algorithm. We then discuss how this algorithm can be used for sampling in PGMs, and how it can be easily accelerated on GPUs.

4.1. Max-product message passing

We consider a PGM with variables z described by a set of A factors $\{\psi_a^\top \phi_a(z^a)\}_{a=1}^A$ and I unary terms $\{\lambda_i^\top \eta_i(z_i)\}_{i=1}^I$. z^a is the vector of variables used in the factor a, ψ_a is a vector of factor parameters and $\phi_a(z^a)$ is a vector of factor sufficient statistics. For the unary terms, the sufficient statistics are the indicator functions $\eta_i(z_i) = (\mathbf{1}(z_i = 0), \mathbf{1}(z_i = 1))$. For a Bayesian network, a factor involving $z^a = (z_a, z_{\mathcal{P}(a)}, z_0)$ is defined for the ath variable. The corresponding ψ_a can be derived from the parameters $\{\theta_{0\to a}\} \cup \{\theta_{k\to a}\}_{k\in\mathcal{P}(a)}$ defined in Equation (1).

The energy of the model can be expressed as $E(z) = -\sum_{a=1}^A \psi_a^\top \phi_a(z^a) - \sum_{i=1}^I \lambda_i^\top \eta_i(z_i)$ or, collecting the parameters and sufficient statistics in corresponding vectors, $E(z) = -\Psi^\top \Phi(z) - \Lambda^\top \eta(z)$. The probability of a configuration z satisfies $p(z) \propto \exp(-E(z))$. The maximum a posteriori (MAP) problem consists in finding the variable assignment with the lowest energy, that is

$$z^{\text{MAP}} \in \operatorname*{argmin}_z E(z) = \operatorname*{argmax}_z \ \Psi^\top \Phi(z) + \Lambda^\top \eta(z) \quad (2)$$

The max-product algorithm estimates this solution by iterating the fixed-point updates for N_{MP} iterations:

$$m_{i\to a}(z_i) = \lambda_i^\top \eta_i(z_i) + \sum_{b\in \text{nb}(i)\setminus a} m_{b\to i}(z_i) \quad (3)$$

$$m_{a\to i}(z_i) = \max_{z_{k\setminus i}} \left\{ \psi_a^\top \phi_a(z^a) + \sum_{k\in\text{nb}(a)\setminus i} m_{k\to a}(z_k) \right\}$$

where $\text{nb}(\cdot)$ denotes the neighbors of a factor or variable. Equations (3) are derived by setting the gradients of the Lagrangian of the Bethe free energy to 0—see Wainwright et al. (2008). $m_{i\to a}(z_i)$ (resp. $m_{a\to i}(z_i)$) are called the "messages" from variables to factors (resp. from factors to variables): max-product is a "message-passing" algorithm. After N_{MP} iterations of Equation (3), max-product estimates the solution to Problem (2) by

$$z_i = \operatorname*{argmax}_c \left\{ \lambda_i^\top \eta_i(z_i = c) + \sum_{b\in\text{nb}(i)} m_{b\to i}(z_i = c) \right\}, \forall i.$$

MP is guaranteed to converge in trees like BNs (Weiss, 1997). A damping factor $\alpha \in (0, 1)$ in the updates can be used to improve convergence, so that $m_{a\to i}^{\text{new}}(z_i) = \alpha m_{a\to i}(z_i) + (1-\alpha) m_{a\to i}^{\text{old}}(z_i)$. $\alpha = 0.5$ offers a good trade-off between accuracy and speed in most cases.

Max-product in BNs: The noisy-OR factor in Equation (1) connects the variables $\{z_i\} \cup \{z_0\} \cup z_{\mathcal{P}(i)}$ and has $2^{2+|\mathcal{P}(i)|}$ valid configurations. At first sight, the max-product updates in Equations (3) have an exponential complexity in $\mathcal{O}(2^{|\mathcal{P}(i)|})$. To scale to large factors, we derive in Appendix A an equivalent representation of this noisy-OR factor for which the updates have a linear complexity $\mathcal{O}(|\mathcal{P}(i)|)$.

4.2. Sampling in PGMs via perturb-and-max-product

In this work, we are interested in answering two types of inference queries in PGMs: MAP queries as in Problem (2) and sampling queries. The perturb-and-MAP framework (Papandreou & Yuille, 2011) unifies these two types of queries by considering the problem:

$$\operatorname*{argmax}_z \left\{ \Psi^\top \Phi(z) + (\Lambda + T\,\varepsilon)^\top \eta(z) \right\} \quad (4)$$

where $\varepsilon \in \mathbb{R}^{2I}$ is a perturbation vector added to the vector of unaries Λ, and T is a noise temperature parameter. When $T = 0$, Problem (4) is the MAP Problem (2). When $T = 1$, Papandreou & Yuille (2011) showed that if the entries of ε are independently drawn from a Gumbel distribution, the solution of Problem (4) approximates a sample from the PGM distribution. Lazaro-Gredilla et al. (2021) recently showed state-of-the-art learning and sampling performance on several PGMs including Ising models and Restricted Boltzmann Machines by using max-product to solve Problem (4). We use their method, named perturb-and-max-product (PMP), in the rest of this paper.

4.3. Accelerating max-product on GPUs

Recently Zhou et al. (2022) open-sourced `PGMax`, a `Python` package to run GPU-accelerated parallel max-product on general factor graphs with discrete variables. The authors showed timing improvements of two to three orders of magnitude compared with alternatives. We use this package to solve the families of perturbed MAP Problems (4) for noisy-OR BNs, while performing GPU-accelerated message updates with linear complexity (see Appendix A).

5. Noisy-OR Bayesian Networks Learning

We now derive a scheme for learning noisy-OR BNs that uses parallel max-product for fast approximate inference.

5.1. Deriving the Elbo

Noisy-OR BNs are directed models with intractable likelihood. Therefore, a standard approach is to maximize the evidence lower bound (Elbo) (Kingma & Welling, 2013):

$$\begin{aligned}\log p(x|\Theta) &\geq \mathbb{E}_{q(h|x,\phi)}\left\{\log p(h,x|\Theta) - \log q(h|x,\phi)\right\} \\ &= \mathbb{E}_{q(h|x,\phi)}\left\{\log p(h,x|\Theta)\right\} + \mathbb{H}\left\{q(h|x,\phi)\right\} \\ &= \mathcal{L}(x,\Theta,\phi), \end{aligned} \tag{5}$$

where $q(h|x,\phi)$ is an approximate posterior, which VI assumes to be the output of a recognition network (Buhai et al., 2020) or a MF posterior (Jaakkola & Jordan, 1999; Ji et al., 2020). The first term in Equation (5) is the expectation of the joint log-likelihood under the approximate posterior distribution, while the second term is the entropy of the approximate posterior. If we set $q(h|x,\phi) = p(h|x,\Theta)$, then the bound in Equation (5) becomes tight. However, the exact posterior of a noisy-OR BN is intractable.

We propose to derive an approximate posterior for a binary observation x as follows. We first use max-product to either **(a)** estimate the mode of the model posterior $\tilde{h}(x, T=0) \approx \operatorname{argmax}_h p(h|x,\Theta)$ or **(b)** get a sample from the model posterior $\tilde{h}(x, T=1) \sim p(h|x,\Theta)$. Similar to Lazaro-Gredilla et al. (2021), we address these posterior queries by clamping the visible variables to their observed value and running max-product, i.e., we set $\lambda_i = (0, -\infty)$ if $x_i = 0$, $\lambda_i = (-\infty, 0)$ if $x_i = 1$ in Problem (4). We then solve Problem (4) with a noise temperature $T = 0$ for **(a)** and $T = 1$ for **(b)** using the PMP method described in Section 4.2. We refer to the posterior inference query **(a)** or **(b)** as:

$$\tilde{h}(x,T) = \texttt{PMP}(x,\ \Theta,\ T). \tag{6}$$

After addressing **(a)** or **(b)**, we define the approximate posterior $q(h|x)$ by a Dirac delta centered at $\tilde{h}(x,T)$: $q(h|x) = \mathbf{1}(h = \tilde{h}(x,T))$. The lower bound in Equation (5) becomes $\mathcal{L}(x,\Theta) = \log p(\tilde{h}(x,T), x \mid \Theta)$. $\mathcal{L}$ does not depend on ϕ, and the entropy of $q(h|x)$ is 0. Let $z = (z_0, \tilde{h}(x,T), x)$. Equation (1) can then be used to decompose the Elbo as a sum over the different factors:

$$\begin{aligned}\mathcal{L}(x,\Theta) = \sum_{i=1}^{m+n} z_i \log\Big(1 - \exp\Big(-\theta_{0\to i} - \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big)\Big) \\ + (1-z_i)\Big(-\theta_{0\to i} - \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big). \end{aligned} \tag{7}$$

5.2. Optimizing the Elbo

The Elbo in Equation (7) admits a closed-form gradient. Let us denote $f(\beta) = \log(1-\exp(-\beta))$ the `log1mexp` function, which we compute accurately using Mächler (2012). Its derivative is $f'(\beta) = \frac{\exp(-\beta)}{1-\exp(-\beta)}$. Let $k \in \mathcal{P}(i)$. Then the partial derivative of the Elbo w.r.t. $\theta_{k\to i}$ is:

$$\frac{\partial \mathcal{L}(x,\Theta)}{\partial \theta_{k\to i}} = z_i z_k f'\Big(\theta_{0\to i} + \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big) + (z_i - 1) z_k \tag{8}$$

A similar relationship holds for $\frac{\partial \mathcal{L}(z,\Theta)}{\partial \theta_{0\to i}}$, by setting $z_0 = 1$.

Parameter sharing: In Sections 6.4 and 6.6, several parent-child pairs (k,i) of the noisy-OR BN use the same parameter θ. The chain rule generalizes the partial derivative w.r.t. θ by summing the right-hand side of Equation (8) over the pairs sharing this parameter.

Stochastic gradients updates: We iterate through the data via mini-batches (Robbins & Monro, 1951), and we form a noisy estimate of the gradient of the Elbo on each mini-batch. This allows (a) scalability of our approach to large datasets (b) escaping local optima. We then use `Adam` (Kingma & Ba, 2014) to update the parameters Θ. Finally, as in Ji et al. (2020), we clip the parameters $\Theta = \max(\Theta, \epsilon)$ to keep the Elbo in Equation (7) finite. Algorithm 1 summarizes one step of parameters updates.

Algorithm 1 Stochastic gradient updates with max-product

Input: Current parameters $\Theta^{(t)}$
Mini-batch $\mathcal{B}^{(t)}$ of size S
Noise temperature T
Learning rate `lr`, Clipping value ϵ
Output: Updated parameters $\Theta^{(t+1)}$
function `UpdateParameters`
  for $x_i \in \mathcal{B}^{(t)}$ **do**
    $\tilde{h}_i(x_i, T) = \texttt{PMP}(x_i,\ \Theta^{(t)},\ T)$ as in Equation (6)
    Compute $\nabla\mathcal{L}(x_i, \Theta^{(t)})$ using Equation (8)
  end for
  $\nabla\mathcal{L}_{\mathcal{B}^{(t)}}(\Theta^{(t)}) = \frac{1}{S}\sum_{x_i\in\mathcal{B}^{(t)}} \mathcal{L}(x_i, \Theta^{(t)})$
  $\Theta^{(t+1)} = \texttt{ADAM}(\Theta^{(t)}, \nabla\mathcal{L}_{\mathcal{B}^{(t)}}(\Theta^{(t)}), \texttt{lr})$
  $\Theta^{(t+1)} = \max(\Theta^{(t+1)}, \epsilon)$
end function

5.3. Robustifying VI using MP

Our objective value differs from the one in Ji et al. (2020). Algorithm 1 optimizes the Elbo defined in Equation (7)—referred to as Elbo^{MP}—w.r.t. the model parameters for a given binary configuration—while Ji et al. (2020) optimize an Elbo derived using MF VI—referred to as Elbo^{VI}. When both are defined, Elbo^{MP} and Elbo^{VI} are two valid lower bounds of the log-likelihood of a noisy-OR BN. Thus, in the rest of this paper, we refer to the Elbo of a method as the maximum of Elbo^{VI} and Elbo^{MP}—Appendix D discusses how we can also define Elbo^{MP} for any VI posterior.

When the approximate posterior is concentrated into a single Dirac delta, Elbo^{MP} is tighter than Elbo^{VI}: Elbo^{VI} is

derived from Elbo^{MP} using Jensen's inequality in Ji et al. (2020, Eq. (6))—see Appendix E.1 for more details. However, the non-zero entropy term present in Elbo^{VI} makes it often tighter when the approximate posterior is not a Dirac delta. While the global optima of Elbo^{VI} would provide a good solution, in practice, the optimization of Elbo^{VI} using the simplistic MF VI posterior is hard and often gets stuck in bad local optima. This explains the catastrophic failures of MF VI in Sections 6.4 and 6.6. In contrast, MP uses a richer posterior, which does not impose a factorized approximation over variables like MF VI, but only relies on the milder MP assumptions (locally consistent marginals, Bethe entropy approximation). This difference in posteriors affects the quality of inference and makes the optimization of Elbo^{MP} easier. As a result, our approach seems better at parameters search. We then propose to robustify MF VI with a hybrid approach, which uses the parameters Θ^{Alg1} learned with Algorithm 1 to initialize the VI training from Ji et al. (2020). This initialization should guide the parameters search of VI and lead to a better optima than standalone VI, while returning a tighter Elbo.

6. Results

We assess the performance of our methods on five categories of binary datasets (a) the `tiny20` dataset discussed in Ji et al. (2020) (b) five large sparse `Tensorflow` datasets, (c) binary matrix factorization datasets (d) seven synthetic datasets introduced in Buhai et al. (2020) (e) the 2D blind deconvolution dataset from Lazaro-Gredilla et al. (2021); and on a synthetic depth dataset derived from MNIST (Deng, 2012). Each experiment is run on a NVIDIA Tesla A100.

6.1. Methods compared

We compare the following methods in our experiments:

- **Full VI**: this is the approach from Ji et al. (2020). The authors did not release their code. To efficiently use their method in our experiments, we re-implemented it in `JAX` (Bradbury et al., 2018), using the variational hyperparameters reported. We use `ADAM` (Kingma & Welling, 2013) as we observe that it leads to better performance than the preconditioning proposed by the authors.
- **Local VI**: This is our re-implementation of the local models proposed by Ji et al. (2020) and described in Section 3, which are required to scale VI to large sparse datasets.
- **MP**: this is the proposed max-product training described in Algorithm 1. Max-product is run with a damping $\alpha = 0.5$ for $N_{\text{MP}} = 100$ iterations. We select the noise temperature $T \in \{0, 1\}$ with better empirical performance.
- **MP + VI**: this is the hybrid training proposed in Section 5.3. We first run Algorithm 1 to learn the parameters Θ^{Alg1}, then run VI training for a few iterations starting from Θ^{Alg1}.

All the methods consider a clipping value $\epsilon = 10^{-5}$ for the parameters Θ. For a given experiment, all the methods use the same learning rate and mini-batch size, and we report the best performance of each method over several initializations—which we describe in Appendix C. We monitor the loss of each method for each experiment to make sure that all the methods have converged.

Method	Num iters	Test Elbo
Full VI	1.5k	-14.41 (0.02)
Full VI	5k	-14.40 (0.02)
Local VI	1.5k	-14.43 (0.02)
Local VI	5k	-14.43 (0.02)
MP (ours)	1k	-14.49 (0.03)
MP + VI (ours)	1.5k	$\mathbf{-14.34}$ (0.02)

Table 1. Test Elbos on the `tiny20` dataset averaged over 10 runs. Higher is better. Our hybrid method outperforms full and local VI.

6.2. Tiny20 dataset

Problem: We first consider the `tiny20` dataset[1] on which Ji et al. (2020) illustrate many of their findings. The dataset contains binary occurrence data for 100 words across $16,242$ postings. As in Ji et al. (2020), we build a three-layers graph with 100 visible and 44 hidden nodes using the procedure in Appendix B. We fix this BN and learn the noisy-OR parameters Θ that maximizes Elbo^{MP} (when learning with MP and Algorithm 1) or Elbo^{VI} (when learning with MF VI as in Ji et al. (2020)).

Training: Our training set consists on 70% of the data at random (i.e. $11,369$ samples). We train full VI and local VI for $1,500$ gradient steps, and for $5,000$ steps. For MP + VI, we first run $1,000$ gradient steps using Algorithm 1 with $T = 0$, then 500 gradient steps using VI. All the methods use full-batch gradients as in Ji et al. (2020) and a learning rate of 0.01.

Results: Table 1 reports the test Elbo (defined in Section 5.3 as the best value between Elbo^{VI} and Elbo^{MP}) of the different methods averaged over 10 random train-test splits. Our hybrid MP + VI approach outperforms all the variational methods by a statistically significant margin. Interestingly, we observe that (a) increasing the number of training iterations slightly improves full and local VI, but it does not make them competitive with our best method; (b) standalone MP is competitive; and (c) as reported in Ji et al. (2020), full VI performs slightly better than local VI, as the latter optimizes a constrained VI objective.

In addition, we note that Ji et al. (2020) reported a lower Elbo of -14.50 for their best full VI method, using 145 nodes (as we do) with a different graph heuristic and a dif-

[1] Accessible at https://cs.nyu.edu/~roweis/data/20news_w100.mat

Dataset	Local VI	MP+VI (ours)
`Abstract`	$-327.19\,(0.05)$	$\mathbf{-324.79}\,(0.05)$
`Agnews`	$-130.90\,(0.07)$	$\mathbf{-126.48}\,(0.02)$
`IMDB`	$-429.54\,(0.02)$	$\mathbf{-428.40}\,(0.01)$
`Patent`	$-578.41\,(0.04)$	$\mathbf{-578.33}\,(0.02)$
`Yelp`	$-294.46\,(0.16)$	$\mathbf{-292.08}\,(0.02)$

Table 2. Test Elbos on the large sparse `Tensorflow` text datasets averaged over 10 runs.

ferent initialization procedure—both not described. Finally, to illustrate the distinction between Elbo^{MP} and Elbo^{VI}, we report these two metrics in Appendix E.2, Table 5. In particular, standalone MP is the best performer for Elbo^{MP}.

6.3. Large sparse Tensorflow text datasets

Datasets: We compare our hybrid method with Ji et al. (2020) on five large sparse `Tensorflow` text datasets (Abadi et al., 2015), which respectively contain scientific documents, news, movie reviews, patent descriptions and Yelp reviews. Note that Ji et al. (2020) only consider two datasets and do not detail their processing procedure. To process each dataset, we first tokenize and vectorize it using a vocabulary size of $10,000$ (removing all the words outside the vocabulary) and a maximum sequence length of 500. Second, we represent each sentence by a binary vector $x \in \{0,1\}^{10,000}$, where $x_j = 1$ if the jth word is present. Our datasets' statistics are in Appendix F.1, Table 7.

Problem: As in Ji et al. (2020), for each dataset, we build a five-layers Bayesian network, and learn the noisy-OR parameters Θ that maximizes Elbo^{MP} or Elbo^{VI}.

Training: We train local VI for $4,000$ gradient steps. For hybrid training, we use $3,600$ steps of Algorithm 1 with $T = 0$, then 400 steps of local VI training. Both methods use a mini-batch size of 128 and a learning rate of 3×10^{-4}.

Results: Table 2 averages the test Elbo of both methods over 10 runs—each run shuffles the training and test set separately. Our hybrid method outperforms local VI on four datasets and is tied on one. In addition, Table 6 in Appendix E.3 compares Elbo^{MP} with Elbo^{VI} on each dataset: the hybrid approach is the best performer for Elbo^{MP} on all the datasets, which shows that hybrid training improves the overall performance of the noisy-OR models.

Timings comparison: Table 8 in Appendix F.2 reports the update times (defined as the average time for one gradient step) of MP and local VI on each dataset: MP is two to four times faster. Despite updating the variational parameters one by one, local VI runs at a reasonable speed as it uses small arrays to represent large sparse datasets. Note that MP runs in parallel and does not exploit the sparsity of the data.

6.4. Binary Matrix Factorization

Problem: Our next problem is Binary Matrix Factorization (BMF). Let n, r, p be three integers with $r < \min(n, p)$ and let $U \in \{0,1\}^{n\times r}, V \in \{0,1\}^{r\times p}$ be two binary matrices. We assume that addition is performed on the Boolean semiring, i.e. $1 + 1 = 1$, and we define a binary matrix $X = UV \in \{0,1\}^{n\times p}$. The BMF problem consists in recovering the binary matrices U and V given the observations X.

This problem is equivalent to learning a noisy-OR BN with p visible nodes and r hidden nodes, and with the positive parameters $\theta^x, \theta^u \in \mathbb{R}_+, \hat{V} \in \mathbb{R}_+^{r\times p}$, such that (a) the failure probability between the ith hidden and the jth visible variable is given by $\exp(-\hat{V}_{ij})$ (b) the prior probability of each hidden variable is equal to $1 - \exp(-\theta^u)$ (c) the noise probability of each visible variable is $1 - \exp(-\theta^x)$. Note that θ^x (resp. θ^u) is shared across all the visible (resp. hidden) variables. Let $\Theta = (\theta^x, \theta^u, \hat{V})$. For $x \in \{0,1\}^p$, the conditional probability of the jth entry x_j is

$$p(x_j = 1 \mid u_1, \dots, u_r, \Theta) = 1 - \exp\Big(-\theta^x - \sum_{i=1}^{r} \hat{V}_{ij} u_i\Big).$$

The rows of X give access to n such observations, and our Algorithm 1 naturally extends to the BMF problem.

Dataset: We fix $n = p$ and consider two increasing sequences of values for $n \in \{100, 200, 400\}$ and for $r/n \in \{0.2, 0.4, 0.6\}$. We additionally fix the probability $p_X = p(X_{ij} = 1) = 0.25,\ \forall i, j$. To do this, we first set $p_{UV} = p(U_{ik} = 1) = p(V_{kj} = 1) = \sqrt{1 - (1 - p_X)^{1/r}}, \forall i, j, k$. We then generate three matrices $V \in \{0,1\}^{r\times p}, U^{\text{train}} \in \{0,1\}^{n\times r}, U^{\text{test}} \in \{0,1\}^{n\times r}$ with prior p_{UV} and define $X^{\text{train}} = U^{\text{train}}V$, $X^{\text{test}} = U^{\text{test}}V$.

Related work: Ravanbakhsh et al. (2016) proposed to learn U and V with max-product by estimating the mode of the joint posterior $\max_{U,V} p(U, V|X)$, using non-symmetric priors for U and V. Their method is very similar to PMP (Lazaro-Gredilla et al., 2021) which proposes to sample from the joint multimodal posterior to solve the 2D blind deconvolution problem, Section 6.6. Both approaches do not consider training and directly solve max-product inference, which cannot be expressed in a mini-batch format and has to run on all the training data simultaneously. These two methods are then memory-intensive, and cannot scale to datasets orders of magnitude larger than the ones used here. In comparison, our MP approach computes the gradient of Elbo^{MP} for each training sample, which is memory-light and allows scaling to larger datasets. We report the results of PMP here, which we accelerate on GPU with `PGMax` (Zhou et al., 2022), and we use p_{UV} as priors for U and V.

Training: We train full VI and BP for $40,000$ gradient steps with batch size 20 and learning rate 0.001. We use

Dataset		Full VI			MP (ours)			PMP
n	r	Test Elbo ↑	Test RE (%) ↓	Update time (s) ↓	Test Elbo ↑	Test RE (%) ↓	Update time (s) ↓	Test RE (%) ↓
100	20	−18.26 (0.76)	**4.32** (0.51)	0.62 (0.03)	**−17.01** (1.28)	4.44 (0.64)	**0.09** (0.00)	9.81 (0.92)
	40	−43.80 (3.27)	**9.15** (1.42)	1.39 (0.05)	**−39.29** (0.90)	9.75 (0.30)	**0.11** (0.00)	9.63 (0.67)
	60	−78.42 (3.18)	10.93 (0.98)	2.56 (0.07)	**−54.25** (1.24)	13.68 (0.53)	**0.13** (0.00)	**7.36** (1.03)
200	40	−103.03 (17.31)	10.80 (2.07)	2.48 (0.05)	**−42.71** (2.07)	**7.05** (0.49)	**0.14** (0.00)	12.57 (0.52)
	80	−295.83 (2.95)	24.98 (0.32)	6.74 (0.07)	**−80.23** (1.68)	11.71 (0.40)	**0.20** (0.14)	**11.05** (0.56)
	120	−362.32 (5.78)	24.55 (0.34)	14.42 (0.42)	**−95.25** (1.46)	13.11 (0.35)	**0.26** (0.15)	**8.11** (0.68)
400	80	—	—	18.49 (0.06)	**−94.95** (2.19)	**9.72** (0.26)	**0.35** (0.00)	12.95 (0.83)
	160	—	—	72.22 (0.30)	**−152.16** (3.21)	**11.06** (0.26)	**0.61** (0.00)	11.74 (0.47)
	240	—	—	167.83 (0.20)	**−154.94** (1.71)	**10.82** (0.27)	**0.86** (0.00)	—

Table 3. BMF results averaged over 10 runs. Arrows pointing up (down) indicate that higher (lower) is better. For large settings, "—" means that we were not able to get the results, due to time-out (for VI) or out-of-GPU-memory error (for PMP).

MP with $T = 1$ to sample from the posterior as it allows to escape local optima during training. For PMP, there is no training and we directly turn to inference using $1,000$ max-product iterations as in Lazaro-Gredilla et al. (2021).

Metrics: We report the Elbo of each method, as well as its update time. We also report its test reconstruction error, which is defined as $\frac{1}{n^2}\|U^{\text{test}}\,\hat{V}^{\text{thre}} - X^{\text{test}}\|_1$, where U^{test} and $\hat{V}^{\text{thre}}$ are binary matrices and we have used $1 + 1 = 1$. $\hat{V}^{\text{thre}}$ is derived by thresholding the learned $\hat{V}$ with a threshold of $\log(2)$: a 1 in $\hat{V}^{\text{thre}}$ corresponds to a failure probability lower than 0.5 in $\hat{V}$. U^{test} is the mode of posterior, estimated as detailed in Appendix D. For PMP, $\hat{V}$ is already binary and we only report its test RE—the update times are not defined for PMP as there is no training.

Results: Table 3 averages the results over 10 runs—each run generate new $V, U^{\text{train}}, U^{\text{test}}$. For $n = 100$, there is no clear winner: MP achieves a higher Elbo, while PMP and VI reach lower REs. However, the performance of VI decreases as n increases: for $n = 200, r \in \{80, 120\}$, VI has a test RE very close to $p_X = 0.25$, which is what would return the trivial estimate $\hat{V} = 0$. In addition, the sequential MF parameters updates make full VI prohibitively slow here: MP is 55 times faster for $n = 200, r = 120$, and 195 times faster for $n = 400, r = 240$. Finally, for $n = 400$, VI did not finish training after three weeks and the test REs are close to p_X, which shows that no latent structure has been recovered. PMP is a solid competitor: it is faster than our method as it has no learning, and it leads to better performance when $r/n = 0.6$. However it cannot scale and runs out of GPU memory for the large $n = 400, r = 240$.

6.5. Overparametrization experiments

Problem: Here, we reproduce the noisy-OR experiment from Buhai et al. (2020). The authors introduced seven synthetic datasets[2]—which we refer to as OVPM. Five datasets (`IMG`, `PLNT`, `UNIF`, `CON8`, `CON24`) are generated from ground truth (GT) noisy-OR BNs while two (`IMG-FLIP` and `IMG-UNIF`) additionally perturb the observations.

The five GT noisy-OR BNs have the same structure, defined as follows. $K^* = 8$ latent variables $u_1, \dots, u_8$, are each associated with a continuous vector of parameters $V^k \in \mathbb{R}^p_+$. $V^1, \dots, V^8$ are shared across all the observations while u_k expresses whether the kth latent variable is active for a given observation. Each latent variable has a prior $1 - \exp(-\theta_k)$ with $\theta_k \geq 0$. Let $\Theta^* = (V^1, \dots, V^8, \theta_1, \dots, \theta_8, \theta^x)$. An observation x is generated such that

$$p(x_j = 1 \mid u_1, \dots, u_r, \Theta^*) = 1 - \exp\left(-\theta^x - \sum_{k=1}^{8} u_k V^k_j\right), \ \forall j.$$

The GT parameters Θ^* are different for each GT noisy-OR BN. Figure 3[left] shows $V^1, \dots, V^8$ and eight cluttered binary samples from one of the datasets, `IMG`[3].

Training: Buhai et al. (2020) learned the noisy-OR BN above for increasing values $K \geq 8$ of latent variables and study how overparametrization improves the recovery of the GT parameters $V^1, \dots, V^8$. We compare our MP approach with their results, using $T = 1$ for MP. For each dataset, we then consider an increasing sequence of latent variables $K \in \{8, 10, 16, 32\}$. We use the same training parameters as Buhai et al. (2020): $9,000$ training samples, 100 epochs, a batch size of 20 and a learning rate of 0.001.

Metrics: We compare the performance of our method with the VI results reported in the main Figure 2 of Buhai et al. (2020, Fig. 2) (the numerical values are in Tables 2 and 5 in their appendices). As the authors, we report the averaged number of GT parameters $V^1, \dots, V^8$ recovered during training—which we compute as detailed in Appendix G.1—and the percentage of runs with full recovery.

Results: Figure 2 compares our method (blue) averaged over 50 repetitions, with VI (orange). Both MP and VI benefit from overparametrization: when K increases, both methods recover more GT parameters. In addition, MP outperforms VI. In particular, for each dataset, for a model with 16 or 32 latent variables, our method (a) always recovers

[2] All the datasets are at https://github.com/clinicalml/overparam

[3] `IMG` is originally from Šingliar & Hauskrecht (2006).

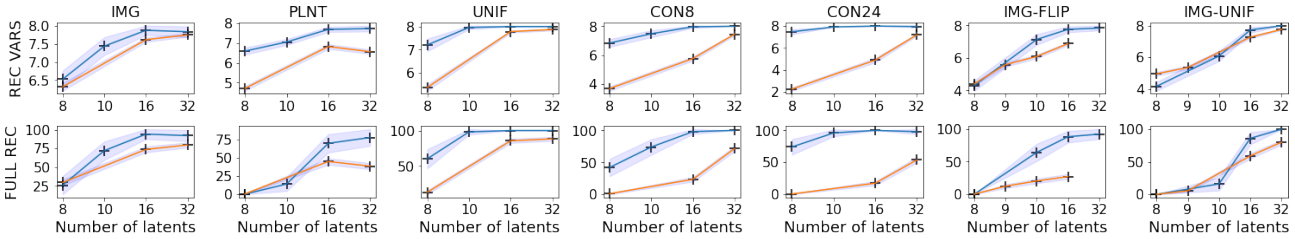

Figure 2. Results for the seven OVPM datasets (a) averaged over 50 repetitions for our method (blue) and (b) reported in Buhai et al. (2020) for VI (orange). As in Buhai et al. (2020), we report the 95% confidence intervals (CIs): the authors considered 500 repetitions, which explain their smaller CIs. Black markers indicate the number of latent variables for which each method is evaluated: Buhai et al. (2020) did not evaluate VI for $K = 10$ on the first five datasets. [Top] Averaged number of GT parameters recovered. [Bottom] Percentage of runs where all the GT parameters are recovered. For overparametrized models with 16 or 32 latent variables, our method always recovers a higher number of GT parameters. All the numerical values are reported in Appendix G.2, Table 9.

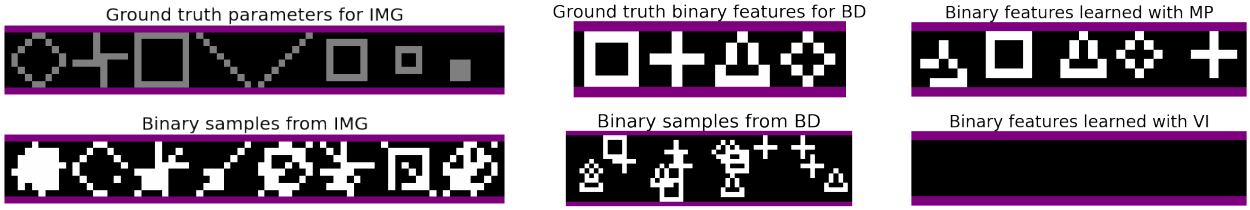

Figure 3. [Top left] Continuous GT parameters for the `IMG` dataset. Grey (resp. black) pixels correspond to failure probabilities of 0.1 (resp. 1.0). [Bottom left] 8 cluttered samples from the `IMG` dataset. [Top middle] GT binary features for the BD problem. [Bottom middle] 4 samples from the BD dataset. [Right] Binary features learned with MP [top] and VI [bottom] for BD for a random seed.

on average at least seven (out of eight) GT parameters (b) always performs better than VI. This gap is larger for the first five datasets, which do not perturb the observations.

6.6. 2D Blind Deconvolution

Problem: Our next experiment is the 2D blind deconvolution (BD) problem from Lazaro-Gredilla et al. (2021, Section 5.6). The task consists in recovering two binary variables W and S from 100 binary images[4] $X \in \{0,1\}^{n \times p}$. W (size: $n_{\text{feat}} \times \text{feat}_{\text{height}} \times \text{feat}_{\text{width}}$) contains 2D binary features. S (size: $n_{\text{images}} \times n_{\text{feat}} \times \text{act}_{\text{height}} \times \text{act}_{\text{width}}$) is a set of binary indicator variables. S and W are combined by convolution, placing the features defined by W at the locations specified by S in order to form X. Unlike S, W is shared by all images. The dimensions of the GT W used to generate X are $4 \times 5 \times 5$, but the authors set the dimensions of the learned $\hat{W}$ to $5 \times 6 \times 6$, which we do too. Figure 3[middle] shows the ground truth W and four samples from X from the BD dataset. Appendix H.1 presents another example from Lazaro-Gredilla et al. (2021), which visualizes S, W and X on a simpler dataset. Note that the BMF experiment, Section 6.5, is a particular case of BD: BD is a harder problem. BD is also equivalent to learning a noisy-OR BN, which we describe in Appendix H.2.

Methods compared: We compare full VI and MP with PMP (discussed in Section 6.4), which directly learns a binary $\hat{W}$ by sampling from the joint posterior $p(S, W|X)$.

Training: We train MP and full VI for $3,000$ steps on 80% of the data, using full-batch gradients and a learning rate of 0.01. PMP has no training and, for inference, we use the same priors as Lazaro-Gredilla et al. (2021).

Metrics: We report the test Elbo, the update time and the test RE of each method. Here, the test RE is defined as $\frac{1}{np}\|X^{\text{test}}_{\text{RE}} - X^{\text{test}}\|_1$, where $X^{\text{test}}_{\text{RE}}$ is computed by convolving the estimated test feature locations S^{test} with the thresholded learned features $\hat{W}^{\text{thre}}$. Finally, we match $\hat{W}^{\text{thre}}$ with the GT W using intersection over union (IOU) for matching and report the features IOU—defined in Appendix H.3. For PMP, we only report the test RE and features IOU.

Results: Table 4 averages the four metrics over 10 repetitions with random train-test splits. VI is 50 times slower than our method and completely fails at recovering the latent structure of the data, which leads to worse test metrics. Again, PMP is a strong competitor. However, (a) it is sensitive to the value of the priors of X, S and W, (b) it leads to a test RE twice higher than MP, (c) it is memory-intensive.

Figure 3[right] shows the binary $\hat{W}^{\text{thre}}$ learned with MP and VI for a random seed—all the results are in Appendix H.4. VI learns continuous features with failure probabilities larger than 0.90 which leads to empty thresholded binary features. MP recovers the four GT features and adds a noisier feature in the first position (which has a smaller prior

[4] To generate the datasets, we use the publicly released code at https://github.com/vicariousinc/perturb_and_max_product

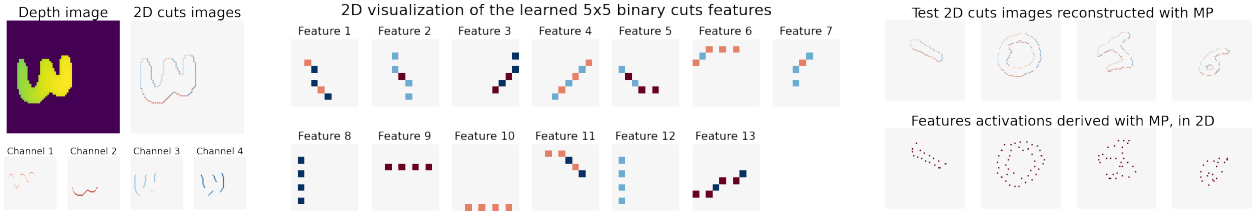

Figure 4. [Top-left] A continuous depth image from our 64×64 augmented `MNIST` dataset and its color-coded cuts activations (each color correspond to a different channel). [Bottom-left] Cuts activations per channel. [Middle] Our 13 learned binary features. [Top-right] Cuts reconstructions on the test scenes. [Bottom-right] Corresponding features activations: each activation is associated with a feature.

and can be easily discarded) while VI fails. Finally, we refer to Appendix H.5 for a comparison of the reconstructed test images returned by our method and PMP.

Metrics	Full VI	PMP	MP (ours)
Test RE (%) ↓	23.46 (0.50)	6.65 (0.79)	**2.96** (0.23)
Features IOU ↑	0.00 (0.00)	0.94 (0.03)	**0.99** (0.01)
Test Elbo ↑	−233.22 (1.33)	N/A	**−43.12**(1.36)
Update time (s) ↓	24.65 (0.67)	N/A	**0.53**(0.00)

Table 4. BD results averaged over 10 runs. Arrows pointing up (down) indicate that higher (lower) is better. VI fails while our method recovers all the features. PMP is a solid competitor.

6.7. Learning binary features from synthetic scenes

Problem: We discuss how the BD formulation can be scaled and extended to learn binary features from synthetic scenes of single objects with perfect depth. To do so, we create a variation of the MNIST dataset (Deng, 2012) that contains $3,994$ continuous depth images Xs of size 64×64: Figure 4[top-left] shows a depth image from our training set. We introduce binary cuts variables $C \in \{0,1\}^{4\times 64\times 64}$ that are active whenever the depth difference between two consecutive pixels is higher than a threshold Δ. That is, we define (a) $C_{1,r,c} = 1$ iff $X_{r,c+1} - X_{r,c} \leq -\Delta$, (b) $C_{2,r,c} = 1$ iff $X_{r,c+1} - X_{r,c} \geq \Delta$, (c) $C_{3,r,c} = 1$ iff $X_{r+1,c} - X_{r,c} \leq -\Delta$, (d) $C_{4,r,c} = 1$ iff $X_{r+1,c} - X_{r,c} \geq \Delta$. As we see in Figure 4[bottom-left], the activations in C capture the contours of the object, and the different channels in C capture the verticality or horizontality of the cuts and their border ownerships. We visualize all the cuts in 2D on a single $V \in \{0,1\}^{128\times 128}$ image where, to display the presence of cuts in-between pixels, we set $V_{2r,2c+1} = \max(C_{1,r,c}, C_{2,r,c})$ and $V_{2r+1,2c} = \max(C_{3,r,c}, C_{4,r,c})$. In particular, $V_{2r,2c} = 0, \forall r, c$. Figure 4 refers to V as the "2D cuts images" and color-codes the channel of each activation.

Learning: For each C, we aim at recovering 16 binary features $W \in \{0,1\}^{16\times 4\times 5\times 5}$ (shared across Cs) and their activations $S \in \{0,1\}^{16\times 60\times 60}$, such that C is obtained by convolving W and S. Each feature in W is now of size $4 \times 5 \times 5$ and has four channels. We train MP as in Section 6.6, using the parameters reported in Appendix C.4.

Results: Figure 4[middle] displays, after thesholding, the 13 learned binary cuts features in 2D: again, the cuts channels are color-coded. The features capture different orientations of the digits: we learn horizontal, vertical and diagonal lines. Note that, as cuts capture border ownership, we learn two vertical and two horizontal features. The test RE is 0.22%: Figure 4[top-right] represents the cuts reconstruction on four test images and Figure 4[bottom-right] represents the inferred activations S in 2D. Each activation is associated with a feature identity, not represented here. Finally, we refer to Appendix I for additional results.

7. Discussion

We have developed a fast, memory-efficient, stochastic algorithm for training noisy-OR BNs using parallel max-product. To scale MP to very large noisy-OR factors, we have turned the max-product updates for a noisy-OR factor from exponential to linear in the number of variables. Contrary to existing VI approaches with a recognition network, our MP method induces explaining-away and recovers more GT parameters on the OVPM datasets. In contrast with MF VI approaches, our method (a) finds better local optima; and (b) scales to large dense datasets. This explains, respectively, why (a) it solves the BD and the BMF problems while MF VI catastrophically fails; and (b) it is up to two orders of magnitude slower. Finally, our method is more memory-efficient than PMP. In addition, we have proposed to use our method to guide VI and help it find better local optima on the large sparse real `Tensorflow` datasets. The main limitation of our method is that max-product requires tractable factors: we cannot apply our algorithm to sigmoid belief networks. Finally, as in the `MNIST` experiment, our next line of work is to use our algorithm to train noisy-OR BNs on visual scenes with complex objects to extract rich latent representations, and build powerful PGMs for vision.

8. Acknowledgments

We thank Wolfgang Lehrach and Théophane Weber for useful discussions during the preparation of this manuscript.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Bishop, C. M. and Nasrabadi, N. M. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Buhai, R.-D., Halpern, Y., Kim, Y., Risteski, A., and Sontag, D. Empirical study of the benefits of overparameterization in learning latent variable models. In *International Conference on Machine Learning*, pp. 1211–1219. PMLR, 2020.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Globerson, A., Chechik, G., Pereira, F., and Tishby, N. Euclidean embedding of co-occurrence data. *Advances in neural information processing systems*, 17, 2004.

Halpern, Y. and Sontag, D. Unsupervised learning of noisy-or bayesian networks. *arXiv preprint arXiv:1309.6834*, 2013.

Jaakkola, T. S. and Jordan, M. I. Variational probabilistic inference and the qmr-dt network. *Journal of artificial intelligence research*, 10:291–322, 1999.

Ji, G., Cheng, D., Ning, H., Yuan, C., Zhou, H., Xiong, L., and Sudderth, E. B. Variational training for large-scale noisy-or bayesian networks. In *Uncertainty in Artificial Intelligence*, pp. 873–882. PMLR, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Koller, D. and Friedman, N. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Lazaro-Gredilla, M., Dedieu, A., and George, D. Perturb-and-max-product: Sampling and learning in discrete energy-based models. *Advances in Neural Information Processing Systems*, 34:928–940, 2021.

Liu, J., Ren, X., Shang, J., Cassidy, T., Voss, C. R., and Han, J. Representing documents via latent keyphrase inference. In *Proceedings of the 25th international conference on World wide web*, pp. 1057–1067, 2016.

Mächler, M. Accurately computing log (1- exp (-— a—)) assessed by the rmpfr package. Technical report, Technical report, 2012.

Murphy, K., Weiss, Y., and Jordan, M. I. Loopy belief propagation for approximate inference: An empirical study. *arXiv preprint arXiv:1301.6725*, 2013.

Papandreou, G. and Yuille, A. L. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision*, pp. 193–200. IEEE, 2011.

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ravanbakhsh, S., Póczos, B., and Greiner, R. Boolean matrix factorization and noisy completion via message passing. In *International Conference on Machine Learning*, pp. 945–954. PMLR, 2016.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Šingliar, T. and Hauskrecht, M. Noisy-or component analysis and its application to link analysis. *The Journal of Machine Learning Research*, 7:2189–2213, 2006.

Wainwright, M. J., Jordan, M. I., et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

Weiss, Y. Belief propagation and revision in networks with loops. 1997.

Zhou, G., Lehrach, W., Dedieu, A., Lázaro-Gredilla, M., and George, D. Graphical models with attention for context-specific independence and an application to perceptual grouping. *arXiv preprint arXiv:2112.03371*, 2021.

Zhou, G., Dedieu, A., Kumar, N., Lázaro-Gredilla, M., Kushagra, S., and George, D. Pgmax: Factor graphs for discrete probabilistic graphical models and loopy belief propagation in jax. *arXiv preprint arXiv:2202.04110*, 2022.

A. Equivalent representation of a noisy-OR factor with optimized messages updates

We discuss herein an equivalent representation of the noisy-OR conditional distribution in Equation (1) that uses the tractable factors supported by `PGMax`.

First, as we have observed in Section 4.1, for the messages from factors to variables, the max-product updates detailed in Equations (3) require to loop through all the valid configurations of a factor. Let i be a variable in the graph, let $N_i = |\mathcal{P}(i)|$ be the cardinality of the set $\mathcal{P}(i)$ of parents of i, and let $\mathcal{P}(i) = \{j_1, \dots, j_{N_i}\}$. The noisy-OR factor associated with i, and described in Equation (1), connects the $2 + N_i$ variables $\{z_i\} \cup \{z_0\} \cup z_{\mathcal{P}(i)}$, and has 2^{2+N_i} valid configuration. Consequently, a naive implementation of the max-product message updates in Equations (3) has a complexity exponential in the number of variables of the noisy-OR factors, which is prohibitively slow for large factors.

To remedy this, let us introduce a family of "noise-free" OR factors which are described by the conditional distribution

$$p(z_i = 0 \mid z_{\mathcal{P}(i)}) = \prod_{k \in \mathcal{P}(i)} (1 - z_k). \tag{9}$$

The noise-free OR factors simply express the logical condition $z_i = \text{OR}(z_{j_1}, \dots, z_{j_{N_i}})$. They do not involve the noisy-OR parameters Θ defined in Equation (1).

It turns out that, for a "noise-free" OR factor, the messages updates derived in `PGMax` have a complexity linear in the number of variables connected to this factor. Consequently, if we derive an equivalent representation of the noisy-OR conditional distribution in Equation (1) that uses the noise-free OR conditional distribution in Equation (9), the cost of the max-product messages updates (using `PGMax`) would go from $\mathcal{O}(2^{|\mathcal{P}(i)|})$ down to $\mathcal{O}(|\mathcal{P}(i)|)$.

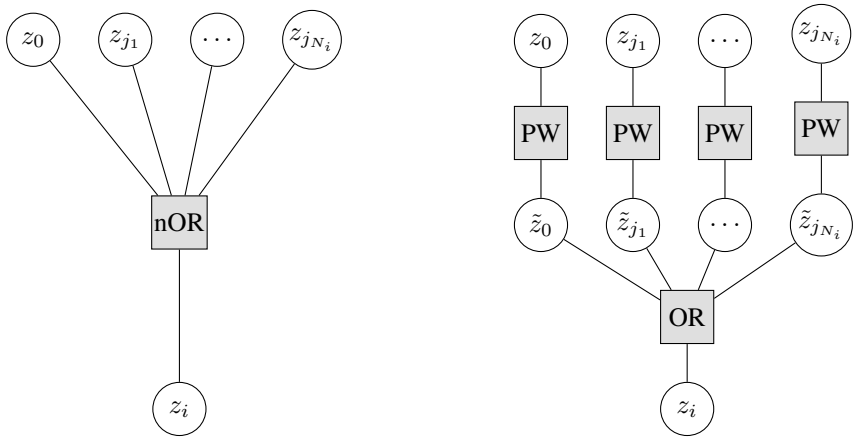

Figure 5. [Left] Noisy-OR factor graph with conditional distribution given by Equation (1). [Right] Equivalent factor graph, which involves a noise-free OR factor with conditional distribution given by Equation (9) and several pairwise factors. For the latter, `PGMax` can perform GPU-accelerated messages updates with a complexity linear in the number of variables connected to the factor.

To this end, we define two factor graphs, which we both represent in Figure 5:

1. The first factor graph considers a single noisy-OR factor—nOR in Figure 5—which connects the leak variable z_0 and the parents variables $z_{\mathcal{P}(i)}$ to the child variable z_i via the noisy-OR conditional distribution defined in Equation (1).

2. The second factor graph introduces the auxiliary binary variables $\tilde{z}_0, \tilde{z}_{j_1}, \dots, \tilde{z_{j_{N_i}}}$ and connects them to the child variable z_i via the noise-free OR factor defined in Equation (9). In addition, for each $k \in \{0\} \cup \mathcal{P}(i)$, there is a pairwise factor—referred to as PW in Figure 5—that connects the variables z_k and $\tilde{z}_k$ and that is defined by

$$\begin{cases} p(\tilde{z}_k = 0 \mid z_k = 0) = 1 \\ p(\tilde{z}_k = 0 \mid z_k = 1) = \exp(-\theta_{k \to i}) \end{cases}$$

   which can be represented in a more compact form by

$$p(\tilde{z}_k = 0 \mid z_k) = \exp(-\theta_{k \to i} z_k). \tag{10}$$

We aim at showing the equivalence between the two factor graphs. To this end, let us derive the conditional distribution of z_i given $z_{\mathcal{P}(i)}$ for the second factor graph:

$$\begin{aligned}
p(z_i = 0 \mid z_{\mathcal{P}(i)}) &= \sum_{\tilde{z}_0 \in \{0,1\},\ \tilde{z}_{\mathcal{P}(i)} \in \{0,1\}^{N_i}} p(z_i = 0,\ \tilde{z}_0,\ \tilde{z}_{\mathcal{P}(i)} \mid z_{\mathcal{P}(i)}) \\
&= \sum_{\tilde{z}_0 \in \{0,1\},\ \tilde{z}_{\mathcal{P}(i)} \in \{0,1\}^{N_i}} p(z_i = 0 \mid \tilde{z}_0, \tilde{z}_{\mathcal{P}(i)})\, p(\tilde{z}_0, \tilde{z}_{\mathcal{P}(i)} \mid z_{\mathcal{P}(i)}) \text{ by conditional independence} \\
&= \sum_{\tilde{z}_0 \in \{0,1\},\ \tilde{z}_{\mathcal{P}(i)} \in \{0,1\}^{N_i}} \prod_{k \in \{0\} \cup \mathcal{P}(i)} (1 - \tilde{z}_k)\, p(\tilde{z}_k \mid z_k) \\
&= \prod_{k \in \{0\} \cup \mathcal{P}(i)} p(\tilde{z}_k = 0 \mid z_k) \text{ as the product cancels if any } \tilde{z}_k = 1 \\
&= \exp(-\theta_{0 \to i}) \prod_{k \in \mathcal{P}(i)} \exp(-\theta_{k \to i} z_k) \text{ using Equation (10) and } z_0 = 1,
\end{aligned}$$

which is exactly the noisy-OR conditional distribution Equation (1). This proves the equivalence between the two factor graphs. In particular, we can use the second factor graph to represent a noisy-OR factor and benefit from the GPU-accelerated messages updates from `PGMax` which have a complexity linear in the number of variables.

B. Graph generation procedure for multi-layered noisy-OR Bayesian networks

We describe below the graph generation procedure we use to build the multi-layered noisy-OR BNs in the `tiny20` and the `Tensorflow` experiments, Sections 6.2 and 6.3.

We assume we are given a binary matrix $X \in \{0,1\}^{n \times p}$, where each row is a binary observation: as in Section 2 there are p visible variables. In the case of the `tiny20` and `Tensorflow` datasets, each visible variable corresponds to a word, and each observation to a sentence or a document: $X_{ij} = 1$ indicates that the jth word is present in the ith document.

Given an integer n_{layers}, we aim at building a noisy-OR Bayesian network with $n_{\text{layers}} + 1$ layers. The bottom layer (with index n_{layers}) of the network contains all the visible nodes, while the trivial top layer (with index 0) only contains the leak node. Our procedure builds the graph iteratively, from the top to the bottom by repeating the two following steps (for j running from n_{layers} down to 1)

1. Build a distance matrix for the jth layer.
2. Create the variables of the $j-1$th layer and add edges connecting the parents of the $j-1$th layer to the children of the jth layer.

We detail these two steps further below.

Distance matrix for the bottom layer: We first detail how we build the distance matrix for the bottom layer—which contains all the visible variables. We start by building a vector of empirical word frequencies $C \in \{0,1\}^p$ such that

$$C_j = \frac{1}{N} \sum_{i=1}^{n} X_{ij},\ \forall j,$$

is the empirical probability that the jth word appears in a document. We also define a matrix of empirical co-occurrence frequencies $O \in \{0,1\}^{p \times p}$ where

$$O_{jk} = \frac{1}{N} \sum_{i=1}^{n} X_{ij} X_{ik},\ \forall j, k,$$

is the empirical probability that the jth and kth words co-occur in a document. From there, we can define the empirical ratio

$$R_{jk} = \frac{O_{jk}}{C_j C_k}.$$

R_{jk} possesses a few interesting properties. First, from the law of large numbers, when n grows to infinity, $C_j \to p(x_j = 1)$, $O_{jk} \to p(x_j = 1, x_k = 1)$ and consequently $R_{jk} \to \frac{p(x_j=1,x_k=1)}{p(x_j=1)p(x_k=1)}$. Therefore, if the jth and kth words are independent, the limit of R_{jk} in the case of an infinite amount of data is 1. If the limit of R_{jk} is higher than 1, then $p(x_j = 1, x_k = 1)$ is higher than the case where the variables are independent. Finally, R_{jk} can also be connected with the mutual information, commonly used in information theory—see Globerson et al. (2004).

Given these properties, we propose to define the distance matrix associated with the bottom layer by

$$D_{jk}^{(n_{\text{layers}})} = \exp(-R_{jk}),\ \forall j, k.$$

Building the $j-1$th layer and connecting it to the jth layer: Let $j \geq 2$. We assume that the jth layer has d variables and that we are given a distance matrix $D^{(j)} \in \mathbb{R}_+^{d\times d}$—we have described above how to build $D^{(n_{\text{layers}})}$ for the bottom layer which has p variables. We now describe how our procedure builds the $j-1$th layer and adds edges between the jth and $j-1$th layers. To this end, we use two hyperparameters: (a) the ratio between the number of variables of the jth layer and of the $j-1$th layer, $r_{\text{children to parents}}$ (which we set to 3 in our experiments) (b) the number of nodes of the $j-1$th layer that each node of the jth layer will be connected with, $n_{\text{parents by node}}$ (which we set to 5).

As a first step, we use hierarchical clustering[5] with average linkage on the distance matrix $D^{(j)}$ to form $\lfloor \frac{d}{r_{\text{children to parents}}} \rfloor$ clusters, with indices $1, \ldots, \lfloor \frac{d}{r_{\text{children to parents}}} \rfloor$. For each cluster m, we then create a variable for the $j-1$th layer, $z_m^{(j-1)}$.

We refer to $\text{label}(z_k^{(j)})$ as the label returned by this clustering step for the kth variable $z_k^{(j)}$ of the jth layer. We could then add edges between the jth and $j-1$th layers by going through the pairs of variables $\left(z_\ell^{(j)}, z_{\text{label}(z_\ell^{(j)})}^{(j-1)}\right)_\ell$. However, if we were doing so, each variable of the jth layer would only be connected to one variable of the $j-1$th layer. The resulting noisy-OR BN would not be able to induce explaining-away (see Section 2) as each effect would be connected to a single cause. To allow inference to induce this appealing property, we propose to add extra edges to the graph by connecting each variable of the jth layer to multiple variables of the $j-1$th layer as follows. First, we define the distance from a variable $z_k^{(j)}$ of the jth layer to a variable $z_m^{(j-1)}$ of the $j-1$th layer as the average distance from $z_k^{(j)}$ to all the elements of the jth layer with label m:

$$\text{dist}(z_k^{(j)}, z_m^{(j-1)}) = \frac{1}{\left|\left\{\ell :\ \text{label}(z_\ell^{(j)}) = m\right\}\right|} \sum_{\ell:\ \text{label}(z_\ell^{(j)})=m} D_{k\ell}^{(j)},\ \forall k, m.$$

Second, we add an edge connecting $z_k^{(j)}$ to the $n_{\text{parents by node}}$ variables of the $j-1$th layer with smallest $\text{dist}(z_k^{(j)}, z_m^{(j-1)})$: this intuitively connects $z_k^{(j)}$ to the $n_{\text{parents by node}}$ labels it is the "closest". Each variable of the jth layer is now connected to the same number of variables of the $j-1$th layer above. However, each variable of the $j-1$th layer may be connected to a different number of variables of the jth layer: we denote $\mathcal{C}(z_m^{(j-1)})$ the set of indices of the variables of the jth layer connected to $z_m^{(j-1)}$. Let us note that, by definition, each node of the $j-1$th and jth layer is also connected to the leak node.

Our last step is to define the symmetric distance matrix $D^{(j-1)}$ between two variables of the $j-1$th layer, which we set to the average distance of all the variables of jth layer connected to these two variables:

$$D_{m_1,m_2}^{(j-1)} = \frac{1}{\left|\mathcal{C}(z_{m_1}^{(j-1)})\right|\ \left|\mathcal{C}(z_{m_2}^{(j-1)})\right|} \sum_{k\in\mathcal{C}(z_{m_1}^{(j-1)})} \sum_{\ell\in\mathcal{C}(z_{m_2}^{(j-1)})} D_{k\ell}^{(j)},\ \forall m_1, m_2.$$

Case j=1: When $j = 1$, as the 0th layer only consists of the leak node, we simply connect each node of the first layer to it.

Remark for the tiny20 graph: We mentioned that, for the `tiny20` experiment, our graph contains 145 nodes and three layers (excluding the top layer). Our graph can be indeed decomposed as follows. The bottom layer contains 100 visible nodes, the second layer contains $\lfloor \frac{100}{3} \rfloor = 33$ hidden nodes, the first layer contains $\lfloor \frac{33}{3} \rfloor = 11$ hidden nodes and the top layer only contains the leak node.

[5] We use the `AgglomerativeClustering` procedure from `scikit-learn` (Pedregosa et al., 2011).

C. Initialization procedures

This section describes the initialization procedures used in the different experiments.

C.1. Tiny20 and large Tensorflow experiments

For each method used in the `tiny20` and the `Tensorflow` experiments, Sections 6.2 and 6.3, we consider the four following initializations for the failure, prior, and noise, probabilities:

1. all the failure probabilities, all the prior probabilities and all the noise probabilities are set to 0.5.
2. all the failure probabilities are set to 0.5, all the prior and noise probabilities are set to 0.1.
3. all the failure probabilities are set to 0.9, all the prior and noise probabilities are set to 0.1.
4. all the failure probabilities are set to 0.9, all the prior and noise probabilities are set to 0.5,

Once we have initialized the aforementioned probabilities, we initialize the parameters Θ accordingly by using the fact that, that for a node i and a node $k \in \mathcal{P}(i)$, the failure probability is $\exp(-\theta_{k\to i})$, while the noise probability—or prior probability when $\mathcal{P}(i)$ is empty—is $p(z_i = 1 \mid z_{\mathcal{P}(i)} = 0,\ \Theta) = 1 - \exp(-\theta_{0\to i})$.

For a given method and a given dataset, we run each initialization for 10 different seeds. We then report the results for the initialization that leads to the best averaged test results.

C.2. BMF and BD experiments

For each method used in the BMF and BD experiments, Sections 6.4 and Sections 6.6, we initialize all the noise probabilities to 0.01 and keep them fixed during training. We have found this to be particularly useful to avoid a local minima where (a) the noise probabilities converge to the average number of activations of the visible variables (b) the prior probabilities converge to 0.

We consider the four following initializations of the remaining failure and prior probabilities:

1. all the failure probabilities and all the prior probabilities are set to 0.5.
2. all the failure probabilities are set to 0.5, all the prior probabilities are set to 0.1.
3. all the failure probabilities are set to 0.9, all the prior probabilities are set to 0.1.
4. all the failure probabilities are set to 0.9, all the prior probabilities are set to 0.5,

In addition, the solution to the BMF and to the BP problems are invariant to certain permutations. For instance, a solution to the BMF problem is invariant to applying the same permutation on the columns of U and the rows of V, while a solution to the BD problem is invariant to applying the same permutation on the features indices (the first dimension) of both W and S. A uniform initialization would then induce symmetries in the parameters during training. To break these symmetries, we add some centered Gaussian noise $\mathcal{N}(0, 0.1)$ to the failure and prior probabilities, before projecting them to $[0, 1]$.

As before, after initializing the failure and prior probabilities (and adding the Gaussian noise), we initialize the parameters Θ accordingly. For a given method and experiment, we run each experiment for 10 different seeds and report the initialization that leads to the best averaged test results.

C.3. OVPM experiment

For the overparametrization experiment, Section 6.5, we follow a very similar procedure to Section C.2, but we only consider the initialization methods 3 and 4, and run each initialization for 50 seeds.

C.4. MNIST experiments

For the feature learning experiment on `MNIST`, Section 6.7, the failure probabilities are set to 0.9 and the prior probabilities are set to 0.1 and the noise probabilities are set to 0.01.

D. Estimating the mode of the model posterior after inference

Given a test sample x, we discuss how to estimate the mode of the model posterior $h^{\text{MAP}} \approx \text{argmax}_h\, p(h|x,\Theta)$ when we use VI and MP at inference time. We use this posterior mode estimation in our experiments to compute (a) Elbo$^{\text{MP}}$ in the `tiny20` and the `Tensorflow` experiments, Sections 6.2 and 6.3, and (b) the test reconstruction errors in the BMF and BD experiments, Sections 6.4 and 6.6.

For MP, we estimate h^{MAP} by clamping the visible variables to their observed value and running max-product with a noise temperature $T = 0$. This is exactly the inference query **(b)** discussed in Section 5.

For VI, the inference from Ji et al. (2020) gives access to the mean-field posterior parameters, that is, the parameters such that, the approximate posterior distribution factorizes as $q(h|x) = \prod_{i\in\mathcal{H}} q_i^{h_i}(1-q_i)^{1-h_i}$. We then estimate the mode of the posterior element-wise via rounding: $h_i^{\text{MAP}} = \mathbf{1}(q_i \geq 0.5),\ \forall i$.

E. Performance comparisons of Elbo$^{\text{MP}}$ and Elbo$^{\text{VI}}$

This section compares Elbo$^{\text{MP}}$ with Elbo$^{\text{VI}}$ for the methods evaluated in the `tiny20` and the `Tensorflow` experiments, Sections 6.2 and 6.3.

1. Elbo$^{\text{VI}}$ is computed by running the inference algorithm of Ji et al. (2020)—which we have reimplemented.
2. To compute Elbo$^{\text{MP}}$, we estimate the posterior mode h^{MAP} as detailed in Section D, then plug it into Equation (7).

E.1. For a binary posterior, Elbo$^{\text{VI}}$ is a lower-bound of Elbo$^{\text{MP}}$

We start by presenting the proof of a claim we made in Section 5.3. We said that, for a binary observation $x \in \{0,1\}^p$, if the posterior $\tilde{h}(x,T)$ is binary, then Elbo$^{\text{VI}}$ is a lower-bound of Elbo$^{\text{MP}}$. To prove this point, let us assume that $\tilde{h}(x,T)$ is binary, let us introduce $z = (z_0, \tilde{h}(x,T), x)$ and let us recall that Elbo$^{\text{MP}}$ is defined in Equation (7) as:

$$\begin{aligned}\mathcal{L}(x,\Theta) &= \sum_{i=1}^{m+n} z_i \log\Big(1 - \exp\Big(-\theta_{0\to i} - \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big)\Big) + (1-z_i)\Big(-\theta_{0\to i} - \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big) \\ &= \sum_{i=1}^{m+n} z_i f\Big(\theta_{0\to i} + \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big) + (1-z_i)\Big(-\theta_{0\to i} - \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big),\end{aligned} \tag{11}$$

where we have used $f(\beta) = \log(1-\exp(-\beta))$. Equation (11) is exactly Equation (3) in Ji et al. (2020) in the case of a binary posterior. From there, as $\theta_{0\to i} \geq 0$ and $\theta_{k\to i} z_k \geq 0,\ \forall k \in \mathcal{P}(i)$, the authors introduced an auxiliary parameter $r_{k\to i}$ for each edge connecting a non-leak parent variable to a child variable such that

$$r_{k\to i} \geq 0,\ \forall k \in \mathcal{P}(i);\ \text{and} \sum_{k\in\mathcal{P}(i)} r_{k\to i} = 1.$$

Consequently $\sum_{k\in\mathcal{P}(i)} r_{k\to i} z_k \in [0,1]$. As f is concave, the authors use Jensen's inequality to get the following lower-bound:

$$\begin{aligned} f\Big(\theta_{0\to i} + \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big) &= f\Big(\Big(1 - \sum_{k\in\mathcal{P}(i)} r_{k\to i} z_k\Big)\theta_{0\to i} + \sum_{k\in\mathcal{P}(i)} r_{k\to i} z_k \Big(\theta_{0\to i} + \frac{\theta_{k\to i}}{r_{k\to i}}\Big)\Big) \\ &\geq \Big(1 - \sum_{k\in\mathcal{P}(i)} r_{k\to i} z_k\Big) f(\theta_{0\to i}) + \sum_{k\in\mathcal{P}(i)} r_{k\to i} z_k f(u_{k\to i}) \text{ where } u_{k\to i} = \theta_{0\to i} + \frac{\theta_{k\to i}}{r_{k\to i}} \\ &= f(\theta_{0\to i}) + \sum_{k\in\mathcal{P}(i)} r_{k\to i} z_k \Big(f(u_{k\to i}) - f(\theta_{0\to i})\Big) \end{aligned} \tag{12}$$

By pairing Equations (11) and (12) we get:

$$\mathcal{L}(x,\Theta) \geq \sum_{i=1}^{m+n} z_i \left\{ f(\theta_{0\to i}) + \sum_{k\in\mathcal{P}(i)} r_{k\to i} z_k \Big(f(u_{k\to i}) - f(\theta_{0\to i})\Big) \right\} + (1-z_i)\Big(-\theta_{0\to i} - \sum_{k\in\mathcal{P}(i)} \theta_{k\to i} z_k\Big). \tag{13}$$

The right-hand size of Equation (13) is exactly Elbo^{VI} in the case of a binary posterior, as defined in Equation (9) in Ji et al. (2020). Consequently, Equation (13) proves that for a binary posterior, Elbo^{MP} is a tighter lower-bound of the intractable log-likelihood of a noisy-OR BN than Elbo^{VI}. Hence, in all our experiments, we never compute Elbo^{VI} for a binary posterior.

E.2. Performance comparisons on the tiny20 dataset

Table 5 reports the averaged test Elbo^{MP} and Elbo^{VI} on the `tiny20` dataset. Standalone MP is trained with Algorithm 1 to optimize Elbo^{MP}. As a result, the MP parameters land in a local optima of this loss and MP reaches the highest test Elbo^{MP}. MP is also the worst performer for Elbo^{VI} as it has not been exposed to this loss during training.

In comparison, all the methods trained with Elbo^{VI} (including the hybrid method MP+VI) perform better at test time for Elbo^{VI} than for Elbo^{MP}. Full VI performs particularly poorly for Elbo^{MP} as it has never been exposed to it during training.

Finally, Table 5 suggests that initializing the VI training with MP helps VI find a better local optima of Elbo^{VI}, which is why our hybrid method reaches the best overall lower bound—while maintaining a high Elbo^{MP}.

Method	Num iters	Test Elbo^{MP}	Test Elbo^{VI}
Full VI	1.5k	-14.80 (0.03)	-14.41 (0.02)
Full VI	5k	-14.85 (0.03)	-14.40 (0.02)
Local VI	1.5k	-14.65 (0.03)	-14.43 (0.02)
Local VI	5k	-14.64 (0.03)	-14.43 (0.02)
MP (ours)	1k	$\mathbf{-14.49}$ (0.03)	-14.49 (0.03)
MP + VI (ours)	1.5k	-14.55(0.02)	$\mathbf{-14.34}$ (0.02)

Table 5. Test Elbo^{MP} and Elbo^{VI} on the `tiny20` dataset averaged over 10 runs.

E.3. Performance comparisons on the large sparse Tensorflow datasets

Table 6 reports the averaged Elbo^{MP} and Elbo^{VI} on the large sparse `Tensorflow` datasets. As before, standalone MP is the worst performer for Elbo^{VI} as it has not been exposed to this loss during training. Local VI is also the worst overall method for Elbo^{MP} for a similar reason. However, it performs better than MP on two datasets, which suggests that, for these datasets, standalone MP is stuck in a local optima during its training.

Our hybrid method is the best performer for both Elbo^{MP} and Elbo^{VI}, which shows that our MP approach finds a good area of the parameters space, that is further refined during the VI optimization of Elbo^{VI}. As a result, the hybrid scheme improves the overall performance of each noisy-OR model.

Dataset	Local VI, Elbo^{MP}	MP, Elbo^{MP}	Hybrid, Elbo^{MP}	Local VI, Elbo^{VI}	MP, Elbo^{VI}	Hybrid, Elbo^{VI}
`Abstract`	-342.79 (0.05)	-342.48 (0.07)	$\mathbf{-327.73}$ (0.06)	-327.19 (0.05)	-335.56 (0.05)	$\mathbf{-324.89}$ (0.05)
`Agnews`	-134.98 (0.07)	-140.48 (0.05)	$\mathbf{-127.75}$ (0.02)	-130.90 (0.07)	-137.88 (0.08)	$\mathbf{-126.48}$ (0.02)
`IMDB`	-450.48 (0.06)	-438.16 (0.04)	$\mathbf{-431.34}$ (0.03)	-429.53 (0.02)	-436.96 (0.04)	$\mathbf{-428.40}$ 0.01
`Patent`	-619.75 (0.07)	-595.91 (0.10)	$\mathbf{-586.08}$ (0.05)	-578.41 (0.04)	-590.33 (0.09)	$\mathbf{-578.33}$ (0.07)
`Yelp`	-303.31 (0.07)	-308.58 (0.09)	$\mathbf{-294.38}$ (0.02)	-294.16 (0.05)	-302.75 (0.07)	$\mathbf{-292.08}$ (0.02)

Table 6. Test Elbo^{MP} and Elbo^{VI} on the large Tensorflow datasets averaged over 10 runs.

F. Additional materials for the large sparse Tensorflow datasets

This section reports some statistics for the large Tensorflow datasets used in Section 6.3, as well as a timing comparison of the different methods used.

F.1. Datasets statistics

For the five large `Tensorflow` datasets, Table 7 below gives access to (a) the full name of the dataset, as it appears in the catalog accessible at https://www.tensorflow.org/datasets/catalog (b) the feature name used when loading the dataset (c) the number of edges in the BNs returned by our graph generation procedure (detailed in Appendix B) (d) the train and test set sizes. In particular, the BNs returned by our procedure have a similar number of edges. This is explained by the fact that, for all the datasets, we use the same number of visible variables—$10,000$—during the preprocessing, and the same hyperparameters during the BN generation.

Dataset	Full name	Feature name	Number of edges	Train set	Test set
`Abstract`	`scientific_papers`	`abstract`	$90,554$	$203,037$	$6,440$
`Agnews`	`ag_news_subset`	`description`	$89,508$	$120,000$	$7,600$
`IMDB`	`imdb_reviews`	`text`	$91,234$	$25,000$	$25,000$
`Patent`	`big_patent/f`	`description`	$90,606$	$85,568$	$4,754$
`Yelp`	`yelp_polarity_reviews`	`text`	$91,111$	$560,000$	$38,000$

Table 7. `Tensorflow` datasets full names and statistics.

F.2. Update times for local VI and MP

Table 8 reports the update time of local VI and MP on the `Tensorflow` datasets, which we have defined in Section 6.3 as the average time for one gradient step. The MP gradients updates detailed in Algorithm 1 run at a very similar speed on all the datasets. Indeed, the complexity of the messages updates is similar across the datasets as (a) as the different BNs have a similar number of edges (as seen in Table 7) (b) MP does not use exploit the sparsity of the data and represents each sentence by a vector $x \in \{0,1\}^{10,000}$.

In contrast, as explained in Section 3, the local models in VI represent each sentence by its active visible variables and by their ancestors. We have set the number of active visible variables per sentence to be at most 500, and in practice it can be lower—some datasets only have a few tenths of active variables on average. Consequently, local VI represents sparse data using arrays three orders of magnitudes smaller than MP. Hence, although local VI updates its variational parameters sequentially, it is reasonably fast. Nonetheless, its update time is dataset-specific and it is two to four times slower than MP.

Dataset	Local VI	MP (ours)
`Abstract`	17.71 (0.05)	$\mathbf{7.51}$ (0.01)
`Agnews`	13.12 (0.07)	$\mathbf{7.39}$ (0.00)
`IMDB`	32.52 (0.11)	$\mathbf{7.51}$ (0.00)
`Patent`	18.72 (0.15)	$\mathbf{7.50}$ (0.00)
`Yelp`	18.89 (0.04)	$\mathbf{7.49}$ (0.00)

Table 8. Update times, in seconds, for local VI and MP on the large sparse `Tensorflow` datasets averaged over 10 runs.

G. Additional material for the overparametrization experiment

This section contains some additional materials for the overparametrization experiment presented in Section 6.5. First, we discuss the method proposed in Buhai et al. (2020) to compute the number of GT parameters recovered during training. Second, we report the table of results associated with Figure 2.

G.1. Computing the number of ground truth parameters recovered

We consider a trained noisy-OR BN with $K \geq 8$ latent variables and learned parameters $\hat{\Theta} = (\hat{V}^1, \dots, \hat{V}^K, \hat{\theta}_1, \dots, \hat{\theta}_K, \hat{\theta}^x)$. We follow the procedure of Buhai et al. (2020) to count the number of recovered GT parameters $V^1, \dots, V^8$—let us trivially note that are at most eight recovered GT parameters.

First, we discard the $\hat{V}^k$ with a prior probability $1 - \exp(-\hat{\theta}_k)$ lower than 0.02. Second, we perform minimum cost bipartite matching between the non-discarded learned parameters and the GT ones $V^1, \dots, V^8$, using the ℓ_{inf} norm as the matching cost. Finally, we count as recovered all the GT parameters with a matching cost lower than 1.0.

G.2. Table of results

Table 9 reports the numerical results of the OVPM experiment which are displayed in Figure 2, Section 6.5. For VI, we take the numbers from Tables 2 and 5 in the appendices of Buhai et al. (2020), which are averaged over 500 repetitions. For MP, our results are averaged over 50 seeds. As in Buhai et al. (2020), we report the 95% confidence intervals of each method.

Dataset		VI		MP (ours)	
Name	Latent variables	Parameters recovered	Full recovery (%)	Parameters recovered	Full recovery (%)
IMG	8	6.31 (0.11)	29.6 (4.0)	**6.52** (0.24)	**26.0** (12.1)
	10	N/A	N/A	7.44 (0.25)	72.0 (12.5)
	16	7.62 (0.06)	73.6 (3.9)	**7.88** (0.13)	**94.0** (6.0)
	32	7.75 (0.05)	79.6 (3.5)	**7.84** (0.15)	**92.0** (7.5)
PLNT	8	4.71 (0.12)	**0.4** (0.6)	**6.60** (0.18)	0.0 (0.0)
	10	N/A	N/A	7.06 (0.15)	14.0 (9.6)
	16	6.83 (0.12)	45.0 (4.4)	**7.70** (0.13)	**70.0** (12.7)
	32	6.57 (0.11)	38.4 (4.3)	**7.74** (0.15)	**78.0** (11.5)
UNIF	8	5.35 (0.14)	12.6 (2.9)	**7.20** (0.27)	**60.0** (13.6)
	10	N/A	N/A	7.96 (0.08)	98.0 (3.9)
	16	7.78 (0.05)	85.4 (3.1)	**8.00** (0.00)	**100.0** (0.0)
	32	7.87 (0.04)	88.2 (2.8)	**8.00** (0.00)	**100.0** (0.0)
CON8	8	3.70 (0.15)	1.2 (1.0)	**6.84** (0.27)	**42.0** (13.7)
	10	N/A	N/A	7.96 (0.08)	98.0 (3.9)
	16	5.77 (0.15)	23.6 (3.7)	**7.96** (0.06)	**98.0** (3.9)
	32	7.45 (0.08)	71.6 (4.0)	**8.00** (0.00)	**100.0** (0.0)
CON24	8	2.26 (0.15)	0.4 (0.6)	**7.48** (0.24)	**74.0** (12.2)
	10	N/A	N/A	7.92 (0.00)	96.0 (5.4)
	16	4.90 (0.21)	17.2 (3.3)	**8.00** (0.00)	**100.0** (0.0)
	32	7.21 (0.10)	53.8 (4.4)	**7.96** (0.08)	**98.0** (3.9)
IMG-FLIP	8	**4.40** (0.10)	**0.2** (0.4)	4.30 (0.32)	0.0 (0.0)
	10	6.09 (0.12)	20.0 (3.5)	**7.14** (0.33)	**64.0** (13.3)
	16	6.88 (0.09)	27.0 (3.9)	**7.76** (0.18)	**88.0** (9.0)
	32	N/A	N/A	7.84 (0.15)	92.0 (13.5)
IMG-UNIF	8	**4.95** (0.12)	0.0 (0.0)	4.16 (0.28)	0.0 (0.0)
	10	N/A	N/A	6.08 (0.32)	16.0 (12.2)
	16	7.27 (0.09)	59.0 (4.3)	**7.72** (0.19)	**86.0** (9.6)
	32	7.76 (0.05)	80.0 (3.5)	**8.00** (0.00)	**100.0** (0.0)

Table 9. Numerical results for the OVPM datasets. We use N/A to express that Buhai et al. (2020) did not evaluate VI for the associated number of latent variables. Our method outperforms VI on all the datasets. In particular, it always recovers more GT parameters in the overparametrized regime, that is for 16 or 32 latent variables. The performance gap is larger for the first five datasets (IMG, PLNT, UNIF, CON8, CON24) for which the data is not perturbed.

H. Additional material for the 2D blind deconvolution experiment

This section contains some additional materials for the 2D blind deconvolution (BD) experiment presented in Section 6.6. First, we discuss a simple example from Lazaro-Gredilla et al. (2021) which illustrates the generative process of the BD dataset. Second, we express the BD problem as a learning problem in a noisy-OR BN. Third, we define the features IOU metric used in Table 4. Finally, we display the continuous and binary features learned by each method, as well as the reconstructed test images for MP and PMP.

H.1. A simple example

Figure 6 uses a simple example from Lazaro-Gredilla et al. (2021) to illustrate the generative process of the BD dataset. The small dataset considered here only contains two independent binary images: each image $X \in \{0,1\}^{15\times 15}$ is formed by convolving the shared binary features $W \in \{0,1\}^{5\times 6\times 6}$ with the image-specific binary locations $S \in \{0,1\}^{5\times 10\times 10}$.

W contains five features, each of size 6×6. S contains the locations of the features, which are sampled at random using an independent Bernoulli prior per entry: $p(S_{f,i,j} = 1) = 0.01,\ \forall f, i, j$. The top (resp. bottom) row of S indicates the locations of the features in the top (resp. bottom) image of X. The jth column of S corresponds to the locations of the jth feature in W. For instance, the two activations on the right of the top-left block of S, means that the first feature in W will appear twice on the right of the first image of X. This is verified by the two anti-diagonal lines in the top row of X.

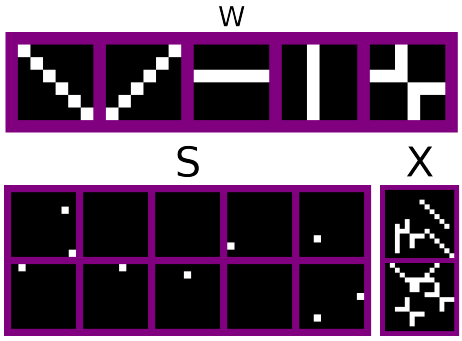

Figure 6. Simple binary convolution example from Lazaro-Gredilla et al. (2021), with features W, locations S and resulting images X.

H.2. The BD problem can be expressed as learning a noisy-OR Bayesian network

The 2D BD problem can be expressed as a learning problem in the noisy-OR BN detailed below.

Let $N \times P$ be the size of an image X. As W is of size $n_{\text{feat}} \times \text{feat}_{\text{height}} \times \text{feat}_{\text{width}}$, S is of size $n_{\text{images}} \times \text{act}_{\text{height}} \times \text{act}_{\text{width}}$, and X is produced from S and W by convolution, let us first note that

$$N = \text{act}_{\text{height}} + \text{feat}_{\text{height}} - 1$$
$$P = \text{act}_{\text{width}} + \text{feat}_{\text{width}} - 1$$

In addition, for a pixel with indices (n, p), let us introduce the set of indices:

$$\mathcal{I}(n,p) = \left\{ (i,j,k,\ell) : \begin{array}{l} 1 \le i \le \text{act}_{\text{height}} \\ 1 \le j \le \text{act}_{\text{width}} \\ 1 \le k \le \text{feat}_{\text{height}} \\ 1 \le \ell \le \text{feat}_{\text{width}} \\ i + k - 1 = n \\ j + \ell - 1 = p \end{array} \right\}.$$

The BD problem is equivalent to learning a noisy-OR BN where (a) the visible nodes are X (b) the hidden nodes are S (c) the positive continuous parameters are $\theta^x \in \mathbb{R}_+$, $\theta_1, \ldots, \theta_{n_{\text{feat}}} \in \mathbb{R}_+$, and $\hat{W} \in \mathbb{R}_+^{n_{\text{feat}} \times \text{feat}_{\text{height}} \times \text{feat}_{\text{width}}}$ and we denote $\Theta = (\theta^x, \theta_1, \ldots, \theta_{n_{\text{feat}}}, \hat{W})$ (d) the prior probability of each entry of S, for the fth feature f is $p(S_{f,i,j} = 1) = 1 - \exp(-\theta_f), \forall i, j$ (e) the conditional probability of the pixel X_{np} is given by

$$p(X_{np} = 1 \mid S, \Theta) = 1 - \exp\left(- \theta^x - \sum_{1 \le f \le n_{\text{feat}}} \sum_{(i,j,k,\ell) \in \mathcal{I}(n,p)} S_{f,i,j} \hat{W}_{f,k,\ell} \right)$$

In particular, the noise probability of each visible variable is equal to $1 - \exp(-\theta^x)$.

H.3. Computing the features intersection-over-union

Let us first define the intersection-over-union (IOU) between a thresholded learned feature $\hat{W}_j^{\text{thre}} \in \{0,1\}^{6\times 6}$ and a GT feature $W_k \in \{0,1\}^{5\times 5}$. To do so, we introduce the four sub-features $\hat{W}_{j,1}^{\text{thre}}, \dots, \hat{W}_{j,4}^{\text{thre}} \in \{0,1\}^{5\times 5}$ of same size as W_k, obtained by removing the first or last row, and the first or last column of $\hat{W}_j^{\text{thre}}$. We then compute:

$$\text{IOU}(\hat{W}_j^{\text{thre}}, W_k) = \max_{\ell=1,\dots,4} \left\{ \frac{\sum_{1 \le n,p \le 5} \text{AND}\Big(\big(\hat{W}_{j,\ell}^{\text{thre}}\big)_{np} = 1, \big(W_k\big)_{np} = 1 \Big)}{\sum_{1 \le n,p \le 5} \text{OR}\Big(\big(\hat{W}_{j,\ell}^{\text{thre}}\big)_{np} = 1, \big(W_k\big)_{np} = 1 \Big)} \right\}.$$

The IOU is always between 0 and 1: IOU $= 0$ means that $\hat{W}_j^{\text{thre}} = 0$ whereas IOU $= 1$ of one means that one of the sub-features $\hat{W}_{j,1}^{\text{thre}}, \dots, \hat{W}_{j,4}^{\text{thre}}$ is equal to W_k.

After training our noisy-OR BN on the BD problem, we perform minimum bipartite matching between the learned binary features $\hat{W}_1^{\text{thre}}, \dots, \hat{W}_5^{\text{thre}}$ and the GT binary features $W_1, \dots, W_4$, using the opposite of the IOU as the matching cost—as we want to maximize the IOU. We then define the features IOU as the average matching cost: a feature IOU of 1 means that we have recovered the four GT features whereas a feature IOU of 0 means that training has not learned any information.

H.4. Learned binary features

Our next Figure 7 plots the five continuous parameters $\hat{W}$ and the corresponding binary features $\hat{W}^{\text{thre}}$ learned by MP, VI, and PMP for each of the 10 seeds. Note that the order of the features is not relevant here, as it depends on the random noise added to the unaries of each model during the initialization—as discussed in Appendix A.

VI completely fails at this task: all the learned failure probabilities are higher than 0.90 and are not visible in Figure 7[third panel]. As a result, all the learned binary features in the fourth panel are empty.

As PMP directly turns to posterior inference, the learned features $\hat{W}$ are binary so we only have one plot. PMP perfectly recovers the four GT features W for seven of the ten runs. It misses two features on one run, and only misses one pixel of one feature on two runs (see fith panel). As the learned $\hat{W}^{\text{thre}}$ contains five features while the GT W only contains four features, each run also learns an extra feature. However, PMP does not provide a way to discard this extra element.

In contrast, as we see on the second panel, MP successfully recovers the four GT features—as well as an extra one—for nine runs, and only misses one pixel of one feature for the other run. This is why MP reaches the highest features IOU in Table 4. The noisy-OR BN trained with MP also learns a prior probability for each feature: the additional feature is always the one with the lowest prior, and can be easily discarded.

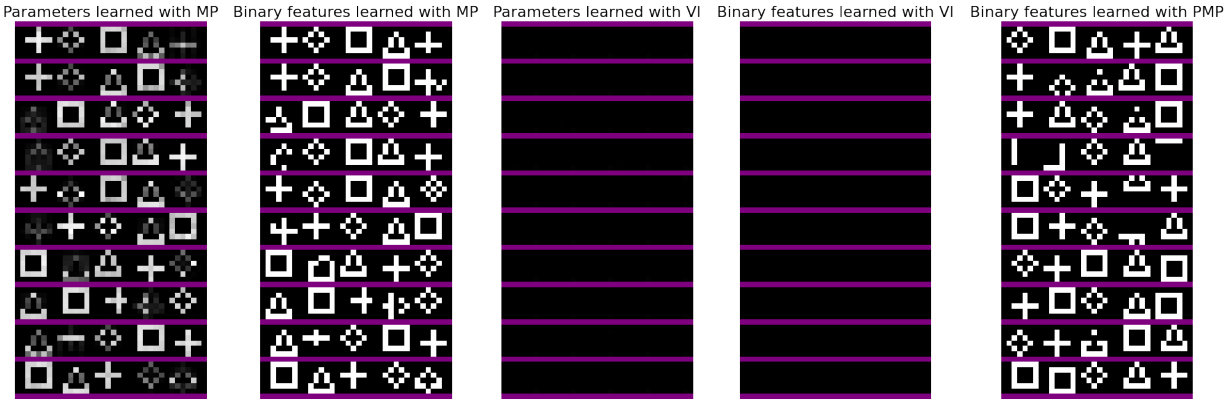

Figure 7. [First panel] Continuous $\hat{W}$ learned with MP. [Second panel] Binary $\hat{W}^{\text{thre}}$ learned with MP. [Third panel] Continuous $\hat{W}$ learned with VI. [Fourth panel] Binary $\hat{W}^{\text{thre}}$ learned with VI. [Fifth panel] Binary $\hat{W}$ learned with PMP.

H.5. Reconstructed test images

Finally, Figure 8 compares the performance of MP and PMP for reconstructing the test scenes on one seed selected at random. We see that PMP performs well, and that our method achieves an almost perfect test reconstruction, which explains that it reaches the lowest test RE in Table 4, Section 6.6.

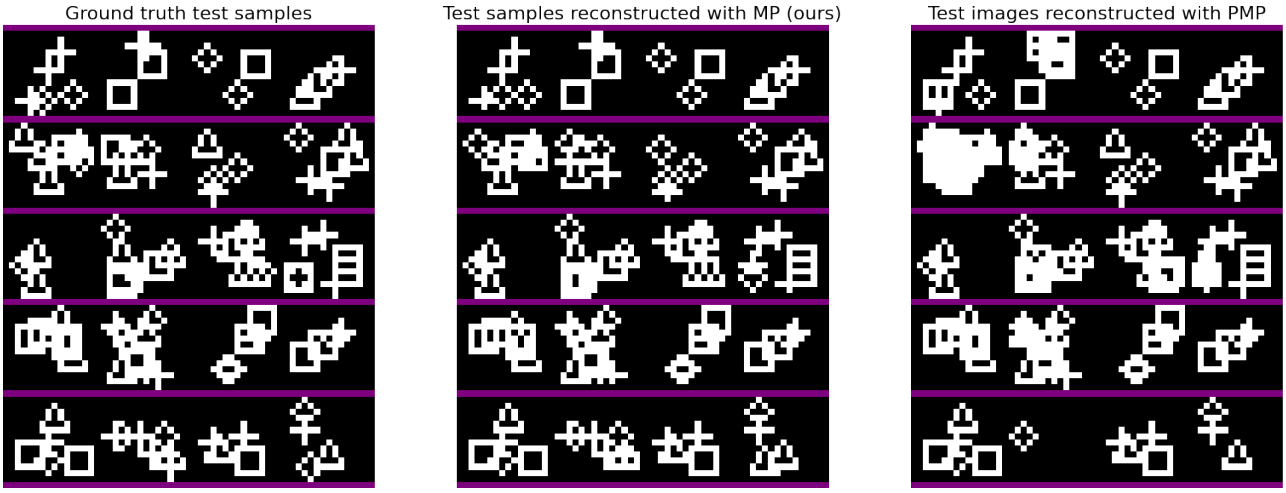

Figure 8. [Left] Ground truth test scenes for a random seed. [Middle] Test reconstructions returned by our MP method. [Right] Test reconstructions returned by PMP.

I. Additional material for the feature learning experiment on synthetic scenes

Our last Figure 9 displays additional depth images from our test set derived from MNIST as well as their respective (a) ground truth 2D cuts images, (b) reconstructed 2D cuts images and (c) activations in 2D. The noisy-OR framework learns sparse representations of contours. It could then be extended to more layers to build a hierarchy of features; or paired with a discrete attention mechanism in PGMs, such as the one proposed in Zhou et al. (2021) for object-agnostic modeling.

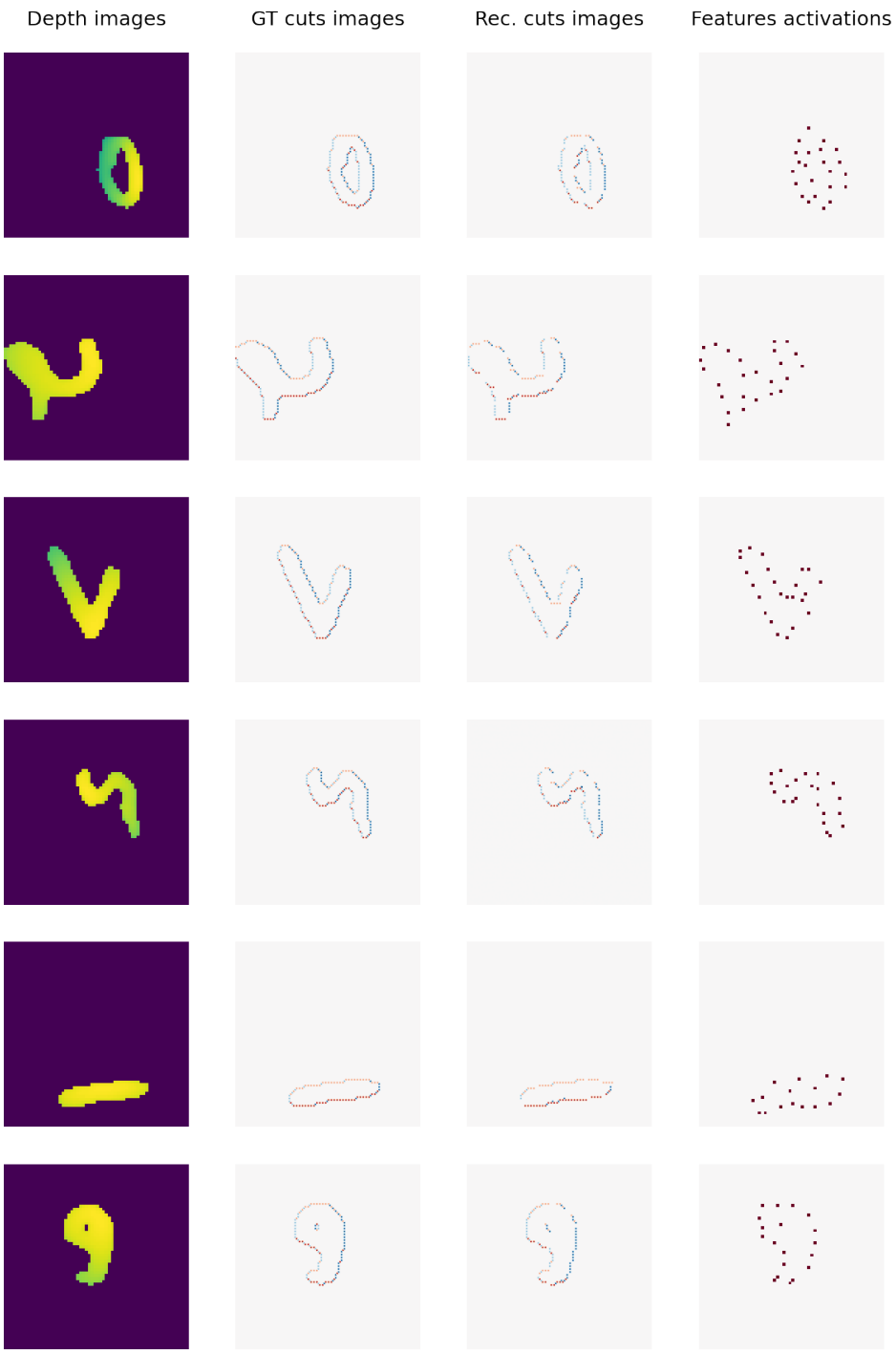

Figure 9. [Left] Six depth scenes from the test set. [Middle left] Ground truth color-coded 2D cuts images. [Middle right] Reconstructed color-coded 2D cuts images. [Right] Sparse features activations, represented in 2D.