A Noisy-OR Networks

A.1 Learning Algorithm

A.1.1 Recognition Network

Recall the evidence lower bound (ELBO):

$$\log p(x;\theta) \geq \mathbb{E}_{q(\cdot|x;\phi)}\left[\log p(x,h;\theta) - \log q(h|x;\phi)\right] = \mathcal{L}(x,\theta,\phi)$$

One can derive the following gradients (see (Mnih & Gregor, 2014)):

$$\nabla_\theta \mathcal{L}(x,\theta,\phi) = \mathbb{E}_{q(\cdot|x;\phi)}[\nabla_\theta \log p(x,h;\theta)]$$

$$\nabla_\phi \mathcal{L}(x,\theta,\phi) = \mathbb{E}_{q(\cdot|x;\phi)}[(\log p(x,h;\theta) - \log q(h|x;\phi)) \times \nabla_\phi \log q(h|x;\phi)]$$

Then, we maximize the lower bound by taking gradient steps w.r.t. θ and ϕ. To estimate the gradients, we average the quantities inside the expectations over multiple samples from $q(\cdot|x;\phi)$.

See Algorithm 1 for an update step of the algorithm, without variance normalization and input-dependent signal centering.

The experiments use mini-batch size 20 (unless specified otherwise). θ and ϕ are optimized using Adam.

Algorithm 1 Update step for learning noisy-OR networks with a recognition network. $p(\cdot,\cdot;\theta)$ is the current noisy-OR network model (with parameters θ), $q(\cdot|\cdot;\phi)$ is the current recognition network model (with parameters ϕ), and x is a batch of 20 samples.

function Update(p, q, x)
  for $i \leftarrow 1$ to 20 **do**
    $h^{(i)} \leftarrow \text{sample}(q(\cdot|x^{(i)};\phi))$
    $r^{(i)} \leftarrow \log p(h^{(i)}, x^{(i)};\theta) - \log q(h^{(i)}|x^{(i)};\phi)$
  end for
  $\Delta\phi \leftarrow \Delta\phi + \eta \cdot \frac{1}{20}\sum_{i=1}^{20} r^{(i)} \cdot \nabla_\phi \log q(h^{(i)}|x^{(i)};\phi)$
  $\Delta\theta \leftarrow \Delta\theta + \eta \cdot \frac{1}{20}\sum_{i=1}^{20} \nabla_\theta \log p(h^{(i)}, x^{(i)};\theta)$
end function

Variance Normalization and Input-Dependent Signal Centering

We use variance normalization and input-dependent signal centering to improve the estimation of $\nabla_\theta \mathcal{L}(x,\theta,\phi)$, as in (Mnih & Gregor, 2014).

The goal of both techniques is to reduce the variance in the estimation of $\nabla_\phi \mathcal{L}(x,\theta,\phi)$. They are based on the observation that (see (Mnih & Gregor, 2014)):

$$\begin{aligned}\nabla_\phi \mathcal{L}(x,\theta,\phi) &= \mathbb{E}_{q(\cdot|x;\phi)}[(\log p(x,h;\theta) - \log q(h|x;\phi)) \times \nabla_\phi \log q(h|x;\phi)] \\ &= \mathbb{E}_{q(\cdot|x;\phi)}[(\log p(x,h;\theta) - \log q(h|x;\phi) - c) \times \nabla_\phi \log q(h|x;\phi)]\end{aligned}$$

where c does not depend on h. Therefore, it is possible to reduce the variance in the estimator by using some c close to $\log p(x,h;\theta) - \log q(h|x;\phi)$.

Variance normalization keeps running averages of the mean and variance of $\log p(x,h;\theta) - \log q(h|x;\phi)$. Let c be the average mean and v be the average variance. Then variance normalization transforms $\log p(x,h;\theta) - \log q(h|x;\phi)$ into $\frac{\log p(x,h;\theta) - \log q(h|x;\phi) - c}{\max(1,\sqrt{v})}$.

Input-dependent signal centering keeps an input-dependent function $c(x)$ that approximates the normalized value of $\log p(x,h;\theta) - \log q(h|x;\phi)$. We model $c(x)$ as a two-layer neural network with 100 hidden nodes in the second layer and tanh activation functions. We train $c(x)$ to minimize

$$\mathbb{E}_{q(\cdot|x;\phi)}\left[\left(\frac{\log p(x,h;\theta) - \log q(h|x;\phi) - c}{\max(1,\sqrt{v})} - c(x)\right)^2\right]$$

and optimize it using SGD.

Therefore, our estimator of $\nabla_\phi \mathcal{L}(x, \theta, \phi)$ is obtained as:

$$\nabla_\phi \mathcal{L}(x, \theta, \phi) \approx \hat{\mathbb{E}}_{q(\cdot|x;\phi)} \left[\left(\frac{\log p(x, h; \theta) - \log q(h|x; \phi) - c}{\max(1, \sqrt{v})} - c(x) \right) \times \nabla_\phi \log q(h|x; \phi) \right]$$

A.1.2 Mean-field Variational Posterior

In the mean-field algorithm, we use the variational posterior $q(h) = \prod_{i=1}^m q_i(h_i)$. That is, the latent variables are modeled as independent Bernoulli.

For each data point, we optimize q from scratch (unlike the case of the recognition network variational posterior, which is "global"), and then we make a gradient update to the generative model.

To optimize the variational posterior q we use coordinate ascent, according to:

$$q_i(h_i) \propto \exp\{\mathbb{E}_{[m]\setminus\{i\}}[\log p(h_i, h_{[m]\setminus\{i\}}, x)]\}$$

where the expectation is over $h_{[m]\setminus\{i\}}$.

See Algorithm 2 for an update step of the algorithm. We use 5 iterations of coordinate ascent, and we use 20 samples to estimate expectations.

Algorithm 2 Update step for learning noisy-OR networks with mean-field variational posterior. $p(\cdot, \cdot; \theta)$ is the current noisy-OR network model (with parameters θ), and x is a sample.

function Update(p, x)
  $q \leftarrow$ independent_Bernoulli_variables_distribution(m)
  for *iter* $\leftarrow 1$ to 5 **do**
    for $k \leftarrow 1$ to m **do**
      $E = [0, 0]$
      for $t \leftarrow 1$ to 20 **do**
        $h_k = 0$
        $h_{[m]\setminus\{k\}} \leftarrow \text{sample}(q(\cdot))$
        $E[0] \leftarrow E[0] + \frac{1}{20} \log p(h, x; \theta)$
        $E[1] \leftarrow E[1] + \frac{1}{20} \log p(h + 1_k, x; \theta)$
      end for
      $q_k(\cdot) \propto \exp\{E[\cdot]\}$
    end for
  end for
  for $i \leftarrow 1$ to 20 **do**
    $h \leftarrow \text{sample}(q(\cdot))$
    $\Delta\theta \leftarrow \Delta\theta + \eta \cdot \frac{1}{20} \cdot \nabla_\theta \log p(h, x; \theta)$
  end for
end function

A.2 Experiment Configuration

Initialization

In all noisy-OR network experiments, the model is initialized by sampling each prior probability π_i, each failure probability f_{ji}, and each complement of noise probability $1 - l_j$ as follows: sample $z \sim \text{uniform}[2.0, 4.0]$, and then set the parameter to $\text{sigmoid}(z)$. Note that $\text{sigmoid}(z)$ is roughly between 0.88 and 0.98, so the noisy-OR network is biased toward having large prior probabilities, large failure probabilities, and small noise probabilities. We found that this biased initialization improves results over one centered around 0.5.

In the recognition network experiments, we initialize the recognition network to have all weight parameters and bias parameters uniform in $[-0.1, 0.1]$.

In the mean-field network experiments, we initialize the mean-field Bernoulli random variables to have parameters uniform in $[0.2, 0.8]$.

Hyperparameters

We generally tune hyperparameters within factors of 10 for each experiment configuration. Due to time constraints, for each choice of hyperparameters we only test it on 10 runs with random initializations of the algorithm, and then choosing the best performing hyperparameters for the large-scale experiments.

In the recognition network experiments, we tune the step size for the noisy-OR network model parameters and the step size for the input-dependent signal centering neural network. The step size for the recognition network model parameters is the same as the one for the noisy-OR network model parameters (tuning it independently did not seem to change the results significantly). The variance reduction technique requires a rate for the running estimates; we set this to 0.8.

In the mean-field experiments, we only tune the step size for the noisy-OR network model parameters. For the coordinate ascent method used to optimize the mean-field parameters, we use 5 iterations of coordinate ascent (i.e. 5 iterations through all coordinates), and in each iteration we estimate expectations with 20 samples.

Number of Epochs

In all experiments, we use a large enough fixed number of epochs such that the log-likelihood does not improve at the end of the optimization in all or nearly-all the runs. However, to avoid overfitting, we save the model parameters at regular intervals in the optimization process, and report the results from the timestep that achieved the best validation set log-likelihood (i.e. we perform post-hoc "early stopping").

B Sparse Coding

B.1 Learning Algorithm

See Algorithm 3 for an update step of the algorithm. We use batch size 20 in the learning algorithm in all experiments.

Algorithm 3 Alternating minimization algorithm update for sparse coding. A is the current matrix, and x is a batch of 20 samples.

function Update(A, x)
  for $i \leftarrow 1$ to 20 **do**
    $h^{(i)} \leftarrow \max(0, A^\dagger x^{(i)} - \alpha)$
  end for
  $A \leftarrow A - \eta \cdot \frac{1}{20} \sum_{i=1}^{20} [(Ah^{(i)} - x^{(i)})h^{(i)T}]$
  normalize columns of A
end function

B.2 Experiment Configuration

Initialization

We initialize the matrix by sampling each entry from a standard Gaussian, and then normalizing the columns such as to have unit l_2 norm.

Hyperparameters

We tune the step size for the updates to the matrix and the α variable. We tune hyperparameters within factors of 10 for each experiment configuration.

Number of Epochs

In all experiments, we use a large enough fixed number of epochs such that the error does not improve at the end of the optimization in all or nearly-all the runs.

C Neural PCFG

C.1 Model Parameterization

We adopt the same parameterization from Kim et al. (2019) and associate an input embedding $\mathbf{w}_N$ for each symbol N on the left side of a rule (i.e. $N \in \{S\} \cup \mathcal{N} \cup \mathcal{P}$) and run a neural network over $\mathbf{w}_N$ to obtain the rule probabilities. Concretely, each rule type π_r is parameterized as follows,

$$\pi_{S \to A} = \frac{\exp(\mathbf{u}_A^\top f_1(\mathbf{w}_S))}{\sum_{A' \in \mathcal{N}} \exp(\mathbf{u}_{A'}^\top f_1(\mathbf{w}_S))},$$
$$\pi_{A \to BC} = \frac{\exp(\mathbf{u}_{BC}^\top \mathbf{w}_A)}{\sum_{B'C' \in \mathcal{M}} \exp(\mathbf{u}_{B'C'}^\top \mathbf{w}_A)},$$
$$\pi_{T \to w} = \frac{\exp(\mathbf{u}_w^\top f_2(\mathbf{w}_T))}{\sum_{w' \in \Sigma} \exp(\mathbf{u}_{w'}^\top f_2(\mathbf{w}_T))},$$

where $\mathcal{M}$ is the product space $(\mathcal{N} \cup \mathcal{P}) \times (\mathcal{N} \cup \mathcal{P})$, and f_1, f_2 are MLPs with two residual layers,

$$f_i(\mathbf{x}) = g_{i,1}(g_{i,2}(\mathbf{W}_i \mathbf{x})),$$
$$g_{i,j}(\mathbf{y}) = \mathrm{ReLU}(\mathbf{V}_{i,j} \, \mathrm{ReLU}(\mathbf{U}_{i,j} \mathbf{y})) + \mathbf{y}.$$

The bias terms for the above expressions (including for the rule probabilities) are omitted for notational brevity.

C.2 Experiment Configuration

We use symbol embedding size of 256 and MLP hidden size (for the residual layers) of 256 for all our neural PCFGs. Training proceeds by gradient-based optimization on the log marginal likelihood with batch size of 1 and a learning rate of 0.001 with the Adam update rule. The maximum gradient norm is sent to be 3. Model parameters are initialized with Xavier-Glorot initialization. We train for 10 epochs and perform early stopping based on validation perplexity.

The data-generating PCFG is trained on the Penn Treebank (PTB), where we lower-case all words, remove punctuation, take the top 10000 frequent words as the vocabulary (words not in the vocabulary are mapped to a special <unk> token), and train on sentences of up to length 20. We use the standard PTB splits for training the data-generating PCFG. Neural PCFGs trained on synthetic data are trained on 5000 or 50000 samples, with validation and test set sizes of 1000 each.

D Noisy-OR Networks: Data Sets Details

D.1 IMG Data Set Properties

In addition to being easy to visualize, the noisy-OR network model of the IMG data set has properties that ensure it is not "too easy" to learn. Specifically, 5 out of the 8 latent variable do not have "anchor" observed variables (i.e. observed variables for which a single failure probability is different from 1.0). Such anchor observed variables are an ingredient of most known provable algorithms for learning noisy-OR networks. More technically, the model requires a subtraction step in the quartet learning approach of (Jernite et al., 2013).

D.2 PLNT Data Set Construction

We learn the PLNT model from the UCI plants data set (Lichman et al., 2013), where each data point represents a plant that grows in North America and the 70 binary features indicate in which states and territories of North America it is found. The data set contains 34,781 data points.

To learn the data set, we use the learning algorithm described in Section A.1.1, with 20 latent variables. We remove all learned latent variables with prior probability less than 0.01. Furthermore, we transform all failure probabilities greater than 0.50 into 1.00. This transformation is necessary to obtain sparse connections

between the latent variables and the observed variables; without it, every latent variable is connected to almost every observed variable, which makes learning difficult. The resulting noisy-OR network model has 8 latent variables. Each latent variable has a prior probability between 0.05 and 0.20. By construction, each failure probability different from 1.00 is between 0.00 and 0.50.

Figure 1 shows a representation of the latent variables learned in the PLNT data set. As observed, the latent variables correspond to neighboring regions in North America.

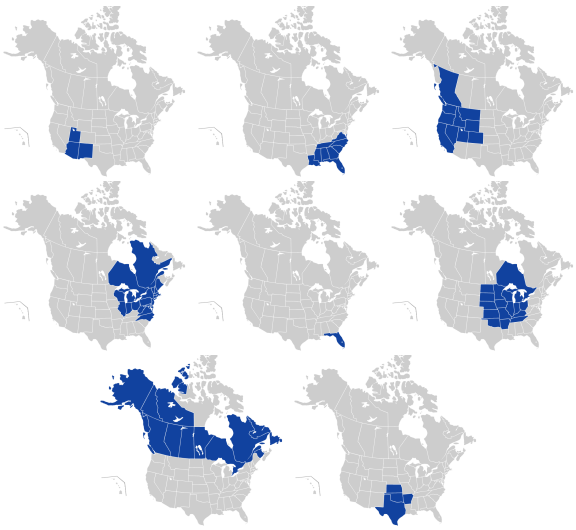


Figure 1: Latent variable configuration of the PLNT noisy-OR network model. Each map represents a latent variable. The regions in blue represent the observed variables for which the failure probability is not 1.00. The fifth latent variable, which seems to contain only Florida, also contains Puerto Rico and Virgin Islands (not shown on map).

E Tables of Results

Below we present detailed tables of results for the experiments.

Table 1 shows the noisy-OR network results with recognition network (partially included in the main document).

Table 2 shows the noisy-OR network results with recognition network and larger batch size 1000.

Table 3 shows the noisy-OR network results with mean-field variational posterior.

Table 4 shows the noisy-OR network results with recognition network on the misspecified data sets (IMG-FLIP and IMG-UNIF).

Table 5 shows the sparse coding results.

Table 6 shows the neural PCFG results (included in the main document).

Table 1: Performance of the noisy-OR network learning algorithm with recognition network. Each row reports statistics for 500 runs of the algorithm with random initializations. The 95% confidence intervals are included. The first column denotes the number of latent variables used in learning, the second column the average number of ground truth latent variables recovered, the third column the percentage of runs with full ground truth recovery, and the fourth column the average test set negative log-likelihood.

LAT VARS	RECOV VARS	FULL RECOV (%)	NEGATIVE LL (NATS)
IMG			
8	6.31 ± 0.11	29.6 ± 4.0	13.76 ± 0.11
16	7.62 ± 0.06	73.6 ± 3.9	12.82 ± 0.05
32	7.75 ± 0.05	79.6 ± 3.5	12.83 ± 0.05
64	7.73 ± 0.05	77.0 ± 3.7	12.95 ± 0.04
128	7.75 ± 0.04	75.6 ± 3.8	12.95 ± 0.04
PLNT			
8	4.71 ± 0.12	0.4 ± 0.6	12.54 ± 0.06
16	6.83 ± 0.12	45.0 ± 4.4	12.05 ± 0.07
32	6.57 ± 0.11	38.6 ± 4.3	12.43 ± 0.08
64	7.03 ± 0.10	55.0 ± 4.4	12.15 ± 0.07
128	7.58 ± 0.08	77.6 ± 3.7	11.79 ± 0.05
UNIF			
8	5.35 ± 0.14	12.6 ± 2.9	11.71 ± 0.13
16	7.78 ± 0.05	85.4 ± 3.1	9.78 ± 0.02
32	7.87 ± 0.04	88.2 ± 2.8	9.72 ± 0.01
64	7.91 ± 0.04	93.6 ± 2.2	9.74 ± 0.01
128	7.78 ± 0.08	91.2 ± 2.5	9.77 ± 0.01
CON8			
8	3.70 ± 0.15	1.2 ± 1.0	8.33 ± 0.10
16	5.77 ± 0.15	23.6 ± 3.7	7.02 ± 0.08
32	7.45 ± 0.08	71.6 ± 4.0	6.08 ± 0.04
64	7.68 ± 0.06	81.6 ± 3.4	5.98 ± 0.03
128	7.88 ± 0.04	92.8 ± 2.3	5.90 ± 0.02
CON24			
8	2.26 ± 0.15	0.4 ± 0.6	14.99 ± 0.17
16	4.90 ± 0.21	17.2 ± 3.3	10.91 ± 0.13
32	7.21 ± 0.10	53.8 ± 4.4	9.76 ± 0.03
64	7.60 ± 0.06	68.4 ± 4.1	9.73 ± 0.02
128	7.34 ± 0.08	52.2 ± 4.4	9.83 ± 0.02

Table 2: Performance of the noisy-OR network learning algorithm with recognition network and batch size 1000. Each row reports statistics for 500 runs of the algorithm with random initializations. The 95% confidence intervals are included. The first column denotes the number of latent variables used in learning, the second column the average number of ground truth latent variables recovered, the third column the percentage of runs with full ground truth recovery, and the fourth column the average test set negative log-likelihood.

LAT VARS	RECOV VARS	FULL RECOV (%)	NEGATIVE LL (NATS)
IMG			
8	5.91 ± 0.13	24.2 ± 3.8	14.09 ± 0.13
16	7.17 ± 0.10	53.6 ± 4.4	13.09 ± 0.09
32	7.32 ± 0.08	58.6 ± 4.3	12.96 ± 0.08
PLNT			
8	4.35 ± 0.12	0.0 ± 0.0	13.95 ± 0.05
16	5.69 ± 0.09	5.0 ± 1.9	13.28 ± 0.05
32	6.18 ± 0.08	15.6 ± 3.2	12.98 ± 0.06
UNIF			
8	5.62 ± 0.17	26.6 ± 3.9	11.27 ± 0.13
16	6.20 ± 0.19	51.0 ± 4.4	10.01 ± 0.04
32	5.84 ± 0.21	49.2 ± 4.4	10.01 ± 0.04
CON8			
8	2.86 ± 0.14	0.0 ± 0.0	8.85 ± 0.10
16	4.31 ± 0.16	6.0 ± 2.1	7.94 ± 0.10
32	7.22 ± 0.10	66.2 ± 4.2	6.21 ± 0.06
CON24			
8	2.52 ± 0.18	2.0 ± 1.2	14.77 ± 0.20
16	5.67 ± 0.24	51.8 ± 4.4	11.68 ± 0.25
32	6.86 ± 0.19	74.4 ± 3.8	10.53 ± 0.19

Table 3: Performance of the noisy-OR network learning algorithm with mean-field variational posterior. Each row reports statistics for 500 runs of the algorithm with random initializations. The 95% confidence intervals are included. The first column denotes the number of latent variables used in learning, the second column the average number of ground truth latent variables recovered, the third column the percentage of runs with full ground truth recovery, and the fourth column the average test set negative log-likelihood.

LAT VARS	RECOV VARS	FULL RECOV (%)	NEGATIVE LL (NATS)
IMG			
8	6.72 ± 0.10	41.8 ± 4.3	13.57 ± 0.13
16	7.52 ± 0.07	66.4 ± 4.1	12.91 ± 0.08
32	7.23 ± 0.09	56.2 ± 4.4	13.12 ± 0.09
PLNT			
8	4.35 ± 0.12	0.0 ± 0.0	12.63 ± 0.08
16	5.69 ± 0.09	5.0 ± 1.9	12.33 ± 0.07
32	6.18 ± 0.08	15.6 ± 3.2	12.32 ± 0.07
UNIF			
8	5.62 ± 0.17	26.6 ± 3.9	11.79 ± 0.12
16	6.20 ± 0.19	51.0 ± 4.4	11.15 ± 0.13
32	5.84 ± 0.21	49.2 ± 4.4	11.65 ± 0.14
CON8			
8	2.86 ± 0.14	0.0 ± 0.0	8.88 ± 0.09
16	4.31 ± 0.16	6.0 ± 2.1	8.34 ± 0.09
32	7.22 ± 0.10	66.2 ± 4.2	8.08 ± 0.09
CON24			
8	2.52 ± 0.18	2.0 ± 1.2	13.27 ± 0.18
16	5.67 ± 0.24	51.8 ± 4.4	12.33 ± 0.19
32	6.86 ± 0.19	74.4 ± 3.8	13.26 ± 0.21

Table 4: Performance of the noisy-OR network learning algorithm with recognition network on the misspecified data sets (IMG-FLIP and IMG-UNIF). Each row reports statistics for 500 runs of the algorithm with random initializations. The 95% confidence intervals are included. The first column denotes the number of latent variables used in learning, the second column the average number of ground truth latent variables recovered, the third column the percentage of runs with full ground truth recovery, and the fourth column the average test set negative log-likelihood.

LAT VARS	RECOV VARS	FULL RECOV (%)	NEGATIVE LL (NATS)
IMG-FLIP			
8	4.40 ± 0.10	0.2 ± 0.4	15.29 ± 0.12
9	5.59 ± 0.13	12.2 ± 2.9	14.46 ± 0.13
10	6.09 ± 0.12	20.0 ± 3.5	13.97 ± 0.11
16	6.88 ± 0.09	27.0 ± 3.9	13.23 ± 0.06
IMG-UNIF			
8	4.95 ± 0.12	0.0 ± 0.0	15.75 ± 0.11
9	5.35 ± 0.12	5.4 ± 2.0	14.87 ± 0.11
16	7.27 ± 0.09	59.0 ± 4.3	13.20 ± 0.05
32	7.76 ± 0.05	80.0 ± 3.5	12.77 ± 0.03

Table 5: Performance of the sparse coding learning algorithm. Each row reports statistics for 500 runs of the algorithm with random initializations. The 95% confidence intervals are included. The first column denotes the number of latent variables used in learning, the second column the average number of ground truth columns recovered, the third column the percentage of runs with full ground truth recovery, and the fourth column the average error.

COLS	RECOV COLS	FULL RECOV (%)	ERROR
$\gamma = 5°$			
24	8.71 ± 0.18	0.0 ± 0.0	9.32 ± 0.18
48	14.54 ± 0.37	1.4 ± 1.0	3.11 ± 0.16
$\gamma = 10°$			
24	19.77 ± 0.29	17.2 ± 3.3	2.28 ± 0.15
48	23.84 ± 0.05	88.4 ± 2.8	0.04 ± 0.02

Table 6: Results from the neural PCFG experiments, where both the training set size and the number of nonterminal/preterminal symbols are varied. The data-generating PCFG is shown at the top, while the learned PCFGs are shown at the bottom. F_1 score is calculated by comparing the MAP parse trees from the learned PCFG against the MAP parse trees from the data-generating PCFG, ignoring the tree labels. For reference, a random tree baseline obtains an unlabeled F_1 score of 30.3.

		5000 SAMPLES		50000 SAMPLES	
$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{P}\rvert$	NEGATIVE LL (NATS)	F_1	NEGATIVE LL (NATS)	F_1
10	10	6.330	–	6.330	–
10	10	6.747	58.1	6.647	69.4
20	20	6.660	62.1	6.582	76.5
30	30	6.658	64.4	6.581	74.0
40	40	6.655	62.0	6.576	72.3

F Noisy-OR Networks: State of the Optimization Process

Figures 2 and 3 show more steps of the optimization process on the successful run with 16 latent variables mentioned in Figure 3.

Figure 2: Latent variables on a successful run of the noisy-OR network learning algorithm on the IMG data set with 16 latent variables. Shown is the state of the latent variables after epochs 1/9, 2/9, 3/9, 4/9, 5/9, 6/9, 8/9, 8/9, and 1.

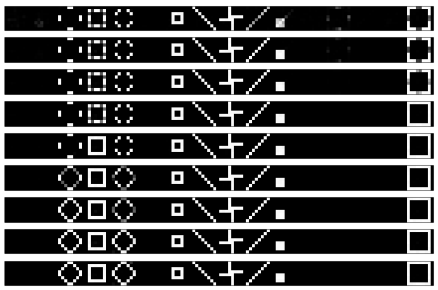
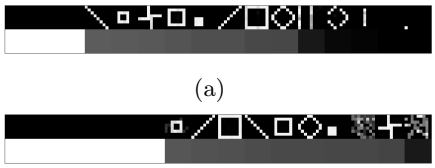


Figure 3: Latent variables on a successful run of the noisy-OR network learning algorithm on the IMG data set with 16 latent variables. Shown is the state of the latent variables after epochs 1, 2, 3, 5, 10, 20, 30, 50, and 100.

G Noisy-OR Networks: Recovered Latent Variables on Misspecified Data Sets

Figure 4 shows successful recoveries for the IMG-FLIP and IMG-UNIF data sets. As observed, some of the extra latent variables are used to model some of the noise due to misspecification.



(a)

(b)

Figure 4: Latent variables recovered in successful runs (i.e. they recover the IMG latent variables) on the mismatched data sets. Below each 8×8 images corresponding to a latent variable, there is a color corresponding to its prior (whiter means prior closer to 1.0). (a) Successful run with 16 latent variables on the IMG-FLIP data set. (b) Successful run with 16 latent variables on the IMG-UNIF data set.

H Noisy-OR Networks: Duplicate Latent Variables Are Not Equivalent to a Single Latent Variable

We give an example of a model with two latent variables that has identical failure probabilities and is not equivalent to a model with one latent variable that has the same failure probabilities (and possibly different priors).

Consider a noisy-OR network model with two latent variables h_1, h_2 and two observed variables x_1, x_2. Let $\pi_1 = \pi_2 = 0.25$ (prior probabilities), $f_{11} = f_{12} = f_{21} = f_{22} = 0.1$ (failure probabilities), and $l_1 = l_2 = 0.0$ (noise probabilities). Then the negative moments are $\mathbb{P}(x_1 = 0) = 0.600625$, $\mathbb{P}(x_2 = 0) = 0.600625$, and $\mathbb{P}(x_1 = 0, x_2 = 0) = 0.56625625$.

Consider now a noisy-OR network model with one latent variable h'_1 and two observed variables x'_1, x'_2. Let $f'_{11} = f'_{21} = 0.1$ and $l'_1 = l'_2 = 0.0$. Then, to match the first-order negative moments $\mathbb{P}(x'_1 = 0) = \mathbb{P}(x_1 = 0)$ and $\mathbb{P}(x'_2 = 0) = \mathbb{P}(x_2 = 0)$, we need $\pi'_1 = 0.44375$ (prior probability). But then this gives $\mathbb{P}(x'_1 = 0, x'_2 = 0) = 0.5606875$, which does not match $\mathbb{P}(x_1 = 0, x_2 = 0)$. Therefore, there exists no noisy-OR model with one latent variable and identical failure and noise probabilities that is equivalent to the noisy-OR model with two latent variables.

I Theoretical Conjecture

Finally, we state some concrete theoretical conjecture that the experiments are indicating:

Conjecture 1 (Overparametrization)**.** *Suppose $p(x, h; \theta) = p(h; \theta)p(x|h; \theta)$ is a single-latent layer generative model, with m latent variables, and n observable variables.*

Then, training an overparametrized model $p(x, h; \tilde{\theta}) = p(h; \tilde{\theta})p(x|h; \tilde{\theta})$ with poly(m, n) latent variables, n observable variables and poly$(m, n, 1/\epsilon)$ training samples, with EM or variational inference with a sufficiently rich class of variational posteriors and random start with high probability recovers a model with parameters $\hat{\theta}$ s.t.

(1) **No overfitting:**

$$KL(p(x, h; \hat{\theta}), p(x, h; \theta)) \leq \epsilon.$$

(2) **Parameter recovery:** *A filtering procedure which removes duplicates (i.e. if nodes h_i, h_j have parameter weights that are close enough, we remove one of them) and low-activity nodes (e.g. $p(h_i = 1)$ is low for binary h, or $\Pr[|h_i| \geq \alpha]$ for real-valued h) results in a set of nodes S, whose parameters are close to the ground truth parameters of $p(x, h; \theta)$.*

References

Jernite, Y., Halpern, Y., and Sontag, D. Discovering hidden variables in noisy-or networks using quartet tests. In *Advances in Neural Information Processing Systems*, pp. 2355–2363, 2013.

Kim, Y., Dyer, C., and Rush, A. M. Compound Probabilistic Context-Free Grammars for Grammar Induction. In *ACL*, 2019.

Lichman, M. et al. Uci machine learning repository, 2013.

Mnih, A. and Gregor, K. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pp. 1791–1799, 2014.