Query Training: Learning a Worse Model to Infer Better Marginals in Undirected Graphical Models with Hidden Variables

Miguel Lázaro-Gredilla, [1] Wolfgang Lehrach, [1] Nishad Gothoskar, [2]
Guangyao Zhou, [1] Antoine Dedieu, [1] Dileep George[1]
[1] Vicarious AI
[2] MIT
{miguel, wolfgang, stannis, antoine, dileep}@vicarious.com, nishadg@mit.edu

Abstract

Probabilistic graphical models (PGMs) provide a compact representation of knowledge that can be queried in a flexible way: after learning the parameters of a graphical model once, new probabilistic queries can be answered at test time without retraining. However, when using undirected PGMS with hidden variables, two sources of error typically compound in all but the simplest models (a) learning error (both computing the partition function and integrating out the hidden variables is intractable); and (b) prediction error (exact inference is also intractable). Here we introduce query training (QT), a mechanism to learn a PGM that is optimized for the approximate inference algorithm that will be paired with it. The resulting PGM is a worse model of the data (as measured by the likelihood), but it is tuned to produce better marginals for a given inference algorithm. Unlike prior works, our approach preserves the querying flexibility of the original PGM: at test time, we can estimate the marginal of any variable given any partial evidence. We demonstrate experimentally that QT can be used to learn a challenging 8-connected grid Markov random field with hidden variables and that it consistently outperforms the state-of-the-art AdVIL when tested on three undirected models across multiple datasets.

Introduction

Probabilistic graphical models (PGMs) define probability density functions (PDFs) over data. Their graph structure describes the conditional dependencies between the random variables of the model, and how the joint PDF can be expressed as a product of factors, with each factor involving only a (typically much smaller) subset of the variables. To fully specify a PGM, in addition to the graphical structure, it is necessary to know the value that each factor takes for each combination of variable assignments. This is typically specified via a set of parameters. Thus, defining a PGM involves choosing a *structure* (typically inspired by the underlying phenomenon being modeled) and estimating its *parameters* (typically chosen to produce the best possible fit to a dataset of interest, according to some fitness criterion, such as maximum likelihood). In this work we assume that the graphical structure has already been specified and concern ourselves with the second part, parameter estimation, also known as learning or training of PGMs.

Generative vs Discriminative PGMs

Generative[1] PGMs model the joint PDF of data, whereas discriminative PGMs model the PDF of a subset of "output" (target) variables given the "input" (evidence) variables.

An advantage of generative PGMs is that we can train the model once on all the data and then, at test time, decide which variables should act as evidence and which variables should act as targets, obtaining valid answers without retraining the model. Furthermore, we can also decide at test time that some variables fall in neither of the previous two categories (unobserved variables), and the model will use the rules of probability to marginalize them out.

This is in contrast with discriminative models, such as multilayer perceptrons (MLPs), in which the set of input and output variables needs to be specified at training time and cannot be changed after the fact without retraining. If there is a need for more than a single evidence/target/unobserved configuration of the variables at test time (query), then a separate MLP needs to be trained for each configuration. In the extreme case in which any query is possible, the number of required MLPs would be exponential in the number of variables of the model, making this approach infeasible. Another disadvantage of the discriminative setup is that each query is trained for independently, despite being related tasks. However, in settings where the query is known a priori and fixed, discriminative models can work well.

Directed vs Undirected PGMs

In the fully observed case, the log-likelihood is a concave function of the natural parameters and its maximum likelihood (ML) estimate should in principle be easy to find. This is indeed the case for directed models, in which each factor can be fit to the data independently, and even has a closed form solution for fully parametric factors. In contrast, undirected PGMs are unnormalized and computing the normalization constant (the partition function) is intractable in general. This partition function depends on the parameters of the PGM, and thus needs to be taken into account during the learning process. The presence of hidden variables further

[1]Literature is not always consistent about the meaning of "generative" models, we follow Ng and Jordan (2002). In particular, when we call a model "generative" we do not imply that there is a simple procedure to sample from it.

complicates the ML learning problem, which is no longer concave and can have multiple local maxima.

The choice between directed and undirected models is usually motivated by the nature of the data. A paradigmatic example of the success of undirected PGMs is the application of Markov random fields (MRFs) to image modeling (Li 2009). The pixels in an image have no natural ordering, so undirected models make the most sense. When the MRF is used to describe a "latent" pixel space from which the actual pixels are generated through a noisy channel, hidden variables emerge naturally. Image modeling also provides an example requiring flexible querying: when inpainting a new image, the subset of pixels to be impainted is only known at test time and cannot be prespecified during training.

Learning in Undirected PGMs

In order to maximize the log-likelihood of a model, the gradients of the partition function (the expectations of the sufficient moments) need to be computed. These could be obtained via Markov chain Monte Carlo (MCMC), but this approach is often too slow for large or complex models.

Practical approaches to learn fully observed undirected models resort instead to fitness functions other than the likelihood, thus sidestepping the computation of the partition function. Relevant examples in the literature are contrastive free energy minimization (Welling and Sutton 2005), piecewise training (Sutton and McCallum 2005), pseudo-moment matching (Wainwright, Jaakkola, and Willsky 2003), or pseudo-likelihood maximization (Besag 1975; Hyvärinen 2006; Sutton and McCallum 2007).

These alternative, tractable fitness functions are sometimes guaranteed to still recover the "correct" (maximum likelihood) parameters of the model as the number of data tends to infinity under some assumptions, but those assumptions rarely hold in practice. For instance, pseudo-moment matching and piece-wise training will recover the correct parameters only if the PGM is a tree (Wainwright, Jaakkola, and Willsky 2003) whereas pseudo-likelihood maximization requires the model space to contain the true generative process of the observed data (almost never the case in practice).

When hidden variables are present, the additional problem of marginalizing them out makes learning even harder (Welling and Sutton 2005; Kuleshov and Ermon 2017; Wiseman and Kim 2019; Li et al. 2019). This is the case that we consider in this work.

Learning a Worse PGM for Better Marginals

As we saw above, practical methods for learning undirected PGMs return parameters that deviate from the ML estimations. Furthermore, since inference is also intractable in general, in practice we query PGMs by resorting to approximate inference. The errors caused by inference are then compounded with the modeling errors. This thought has motivated researchers to embrace the idea of actually learning an admittedly "wrong" PGM, but design the learning scheme to compensate for the inaccuracy of approximate inference, such that the two cancel each other out (Stoyanov, Ropson, and Eisner 2011). This is e.g. the motivation behind pseudo-moment matching as described in Wainwright (2006). In this way, we learn a worse PGM (as compared with the ML estimation) in the hope of obtaining a better approximations for the marginals. Obviously, this "worse PGM" must be paired with a concrete inference scheme. If we were to run a sophisticated MCMC sampler to evaluate the test likelihood of the model, the results would likely be catastrophic.

Even though the idea of learning an appropriately wrong model is appealing, it is still missing information about how we are going to query the model. To illustrate this, consider the approach in Wainwright (2006). The author chooses the "wrong" parameters in such a way that when inference is applied *in the absence of additional evidence*, the right marginals are retrieved. While that specific choice of query (no evidence) seems pretty neutral, it is unlikely to reflect how the model is used in practice. For instance, if all queries were to provide evidence for all the variables minus a subset that is connected forming a tree (different for each query), we know that using the correct parameters is what would result in exact marginals. Conversely, using the correct parameters would fail to provide the right marginals for their selected "no evidence" query when the model is loopy.

Thus, to learn the parameters of a PGM in a way that compensates for the inaccuracies of inference, one needs to take into account how the model is going to be queried at test time, i.e., use a *query distribution*. The expectation is that even a conservative choice, such as a uniform density over all queries, will already generalize better to new queries than optimizing for a single one as in previous works.

Our Approach

Our approach can be described simply at a high level: choose an inference algorithm that can perform marginal inference in the presence of unobserved variables, such as loopy belief propagation (LBP), choose a query distribution, and choose the parameters that maximize the accuracy of the inference algorithm under the query distribution. Because LBP can be unrolled over time into a feed-forward network, parameter optimization can be performed easily using automatic differentiation and stochastic gradient descent. Tools like PyTorch (Paszke et al. 2019) and TensorFlow (Abadi et al. 2015) make gradient computation straightforward.

Prior work unrolling inference into a neural network (NN) includes Domke (2011); Yoon et al. (2019); Lin et al. (2015). However, none of these approaches consider the flexible querying that PGMs are expected to exhibit, and instead fix a single set of inputs and outputs, so that the resulting machine operates in effect as a NN. The cited approaches use PGMs as a motivation for the NN structure, but once the NN is created, the PGM capabilities are not retained.

In contrast, we propose to unroll LBP into a NN that takes as input both the data and a mask describing the query. At training time, that mask will be drawn from the query distribution, randomly attributing variables to the roles of evidence or target, whereas at test time it is given by the query.

Our goal is to provide a systematic framework that turns any PGM into a single NN that (a) handles undirected PGMs with hidden variables; (b) supports flexible *marginal* querying without retraining; and (c) provides a simple mechanism for training and inference.

Query Training (QT)

Our approach takes an (untrained) PGM and unrolls it into a single NN to answer arbitrary queries. It then trains the weights of the NN using different types of queries. The resulting NN can be used at test time for flexible querying, as if it was a PGM. We call this approach query training (QT).

Queries that Need to Be Answered

Given a knowledge representation of some type (e.g. PGM) for the set of variables $\boldsymbol{x} = \{x_1, \ldots, x_N\}$, we want to be able to compute conditional marginal probabilistic queries of the form

$$p(x_{\text{target}}|\{x_i\}_{i\in\text{evidence}}) \quad (1)$$
$$\forall \text{ target} \in \{1, \ldots, N\},\ \forall \text{ evidence} \subset \{1, \ldots, N\}$$

where x_{target} is a single variable, and "evidence" is the subset of the remaining variables for which evidence is available. Any variables that do not correspond to the input nor the output are marginalized out in the above query.

Queries which do not fit Eq. (1) directly (e.g., the joint distribution of two output variables) could be decomposed into a combination of conditional marginal queries by using the chain rule of probability. Though we do not pursue this decomposition here, we want to emphasize that a system that is able to answer queries like Eq. (1) contains enough information to resolve any probabilistic query, with the number of queries being linear in the number of output variables[2]. Because of this, QT is designed to answer marginal queries like Eq. (1) only. In order to have a single system answer arbitrary queries, we follow approximate inference in PGMs.

Approximate Inference in PGMs

Probabilistic queries in PGMs are in most cases intractable, so approximations such as loopy belief propagation (LBP, Murphy, Weiss, and Jordan) or variational inference (VI, Blei, Kucukelbir, and McAuliffe) are used. These approximations are invariant to scale, so the computation of the partition function is *not needed*.

LBP and VI solve an optimization problem iteratively, which is slow. To speed up the process, amortized inference can be used. A prime example of this are VAEs (Kingma and Welling 2013): a learned function (typically an NN) is combined with the reparameterization trick (Rezende, Mohamed, and Wierstra 2014; Titsias and Lázaro-Gredilla 2014) to compute the posterior over the hidden variables given the visible ones. Although a VAE performs inference faster than VI optimization, a single predefined query can be solved. In contrast, LBP and VI answer arbitrary queries (albeit with more compute). Note that standard VAEs can only handle directed PGMs, and a sophisticated variational apparatus with multiple NNs (Li et al. 2019) is required for undirected PGMs.

LBP and VI are closely connected, but behave differently. For instance, for tree PGMs, parallel LBP converges to the exact solution in a number of steps equal to the diameter of the tree (Murphy, Weiss, and Jordan 2013), whereas VI will in general take much longer to converge. In general, LBP tends to produce higher quality marginals in less iterations, so we will choose it over VI in this work.

An Intuitive Description of Query Training

LBP gives a recipe to compute any marginal query of the form of Eq. (1), while sharing the same parameters for all queries. We can then consider LBP, unrolled over a fixed set of iterations, as our inference NN. This NN takes an additional input specifying the desired query. Instead of starting with a trained PGM and generating the inference NN (which is certainly possible), we could generate a "blank" inference NN from an untrained PGM and then learn its (single set of) parameters by minimizing the cross-entropy (CE) between data and its predictions, averaged over the query distribution. We expect this to generalize to new data points and new queries never seen at training time. The intuition behind the existence of a single NN parameterization that approximately satisfies all the queries comes from the good results of running LBP on a correct PGM, which uses a single set of parameters. Note that the the same set of parameters (weights) are shared across all inference steps (layers).

Minimizing the CE between the training data and the query predictions with respect to the model parameters is *not* equivalent to maximum likelihood learning.

The computation of the partition function is sidestepped, so undirected PGMs can be used just as easily: normalization is only required for the output variable of each query, i.e., in one dimension.

LBP is in general only approximate, so one cannot expect predictions to be exact or necessarily consistent (e.g., the query product $p(x = 0|y = 1)p(y = 1)$ is not guaranteed to be identical to the query product $p(y = 1|x = 0)p(x = 0)$, although the cost function will tend to make both similar). On the plus side, the training is free to select the parameters that produce the most precise inference over the training queries, as opposed to the best fit to the data in the ML sense, so this "worse" PGM can compensate for shortcomings in the LBP approximation resulting in better marginals.

Training the Inference Network

The starting point is an unnormalized PGM parameterized by θ. Its probability density can be expressed as $p(\boldsymbol{x}; \theta) = p(\boldsymbol{v}, \boldsymbol{h}; \theta) \propto \exp(\phi(\boldsymbol{v}, \boldsymbol{h}; \theta))$, where $\boldsymbol{v}$ are the visible variables available in our data and $\boldsymbol{h}$ are the hidden variables. A query is a binary vector $\boldsymbol{q}$ of the same dimension as $\boldsymbol{v}$ that partitions the visible variables in two subsets: one for which (possibly soft) evidence is available (inputs) and another whose marginal probability we want to estimate (outputs). Note that we want to compute the marginal of each selected output given all the inputs, independently, that is, for $\mathcal{S} = \{i : q_i = 1\}$, we want to compute all the marginal queries $p(x_i|\{x_j\}_{j\in\mathcal{S}}),\ \forall i \notin \mathcal{S}$.

The query-trained neural network (QT-NN) follows from specifying a graphical model $\phi(\boldsymbol{v}, \boldsymbol{h}; \theta)$, a temperature T and a number of inference timesteps N over which to run

[2]If the answers to these queries is approximate (as it will be the case with QT), different factorizations of a joint density will result in different approximations.

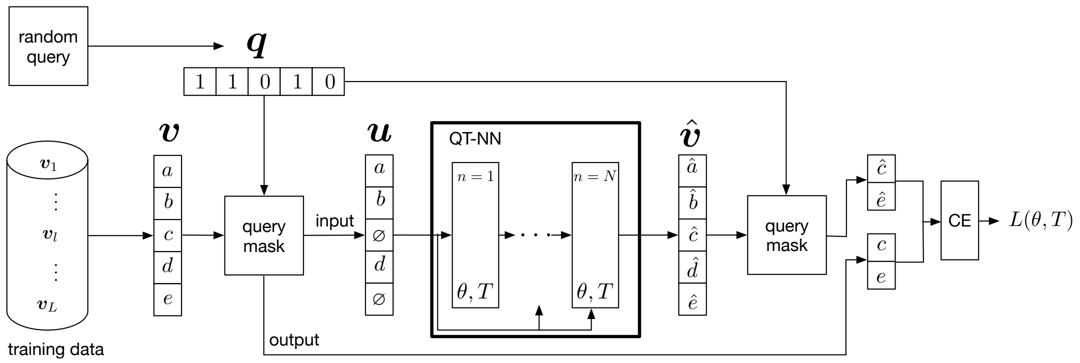

Figure 1: One step of query training. A random sample from the training data is split according to a random query mask in input and output dimensions. The input is processed inside the QT by N identical stages, producing an estimation of the marginal probabilities from the sample. The loss function considers the cross-entropy between the true and estimated marginals. Training uses stochastic samples from both training data and training queries.

parallel BP. The general equations of the QT-NN are given in the next Subsection.

In Fig. 1, a QT-NN takes as input a sample $\boldsymbol{v}$ from the dataset and a random query mask $\boldsymbol{q}$. The query $\boldsymbol{q}$ blocks the network from accessing the "output" variables, and instead only offers access to the "input" variables. The variables assigned to inputs and and outputs change with each query $\boldsymbol{q}$ drawn. The QT-NN produces as output an estimate of the marginal probabilities $\hat{\boldsymbol{v}}$ for the whole input sample. Obviously, we only care about how well the network estimates the variables that it did not see at the input. So we measure how well $\hat{\boldsymbol{v}}$ matches the correct $\boldsymbol{v}$ in terms of cross-entropy (CE), but only for the variables that $\boldsymbol{q}$ regards as "output".

Taking expectation wrt $\boldsymbol{v}$ and $\boldsymbol{q}$, we get $L(\theta, T) = \mathbb{E}_{\boldsymbol{v},\boldsymbol{q}}[\mathrm{CE}_{\boldsymbol{q}}(\boldsymbol{v}, \hat{\boldsymbol{v}})]$, the loss function that we use to train the QT-NN, where the estimated visible units are given by $\hat{\boldsymbol{v}} = \text{QT-NN}(\boldsymbol{v}, \boldsymbol{q}; \theta, T)$.

We minimize this loss wrt θ, T via SGD, sampling from the training data and some query distribution. The query distribution should follow our expected use of the system at test time, and can be set to uniform by default. The number of QT-NN layers N is fixed a priori.

One can think of the QT-NN as a more flexible version of the encoder in a VAE: instead of hardcoding inference for a single query (normally, hidden variables given visible variables), the QT-NN also takes as input a mask $\boldsymbol{q}$ specifying which variables are observed, and provides inference results for unobserved ones. Note that $\boldsymbol{h}$ is never observed, and instead approximately marginalized over by BP.

Unrolling BP into a QT-NN

For a given set of graphical model parameters θ and temperature T we derive a feed-forward function that approximately resolves arbitrary inference queries by unrolling the parallel BP equations for N iterations. First, we combine the available evidence $\boldsymbol{v}$ and the query $\boldsymbol{q}$ into a set of unary factors. Unary factors specify a probability density function over a variable. Therefore, for each dimension inside $\boldsymbol{v}$ that $\boldsymbol{q}$ labels as "input", we provide a (Dirac or Kronecker) delta centered at the value of that dimension. For the "output" dimensions and hidden variables $\boldsymbol{h}$ we set the unary factor to an uninformative, uniform density. Finally, soft evidence, if present, can be incorporated through the appropriate density function. The result of this process is a unary vector of factors $\boldsymbol{u}$ that only contains informative densities about the inputs and whose dimensionality is the sum of the dimensionalities of $\boldsymbol{v}$ and $\boldsymbol{h}$. Each dimension of $\boldsymbol{u}$ will be a real number for binary variables (which we encode in the logit space), and a full distribution in the general case.

Once $\boldsymbol{v}$ and the query $\boldsymbol{q}$ are encoded in $\boldsymbol{u}$, we can unroll parallel LBP over iterations as an NN with N layers, i.e., the QT-NN. To simplify notation, let us consider a PGM that contains only pairwise factors. By mapping the messages to the log-space, the predictions of the QT-NN and the messages from each layer to the next can be written as

$$m_{ij}^{(0)} = 0 \quad m_{ij}^{(n)} = f_{\theta_{ij}}\left(\theta_i + u_i + \sum_{k \neq j} m_{ki}^{(n-1)}; T\right)$$

$$\hat{\boldsymbol{v}}_i = \operatorname{softmax}\left(\theta_i + u_i + \sum_k m_{ki}^{(N)}\right)$$

$$\boldsymbol{m}^{(0)} = 0 \quad \boldsymbol{m}^{(n)} = f_\theta(\boldsymbol{m}^{(n-1)}, \boldsymbol{u}; T) \quad \hat{\boldsymbol{v}} = g_\theta(\boldsymbol{m}^{(N)}, \boldsymbol{u}),$$

where the last row express the same equations as the other two, but in vectorized format.

Here $\boldsymbol{m}^{(n)}$ collects all the messages[3] that exit layer $n-1$ and enter layer n. Messages have direction, so $m_{ij}^{(n)}$ is different from $m_{ji}^{(n)}$[4]. Observe how the input term $\boldsymbol{u}$ is re-fed at every layer. The output of the network is a belief $\hat{\boldsymbol{v}}_i$ for each variable i, which is obtained by a softmax in the last layer. These equations simply correspond to unrolling LBP over iterations, with its messages encoded in log-space.

The portion of the parameters θ relevant to the factor between variables i and j is represented by $\theta_{ij} = \theta_{ji}$, and the

[3]For a fully connected graph, the number of messages is quadratic in the number of variables, showing the advantage of a sparse connectivity pattern, which can be easily encoded in the PGM architecture.

[4]In the general case, $m_{ij}^{(n)}$ and θ_i are vectors. We only encode matrix in bold, to represent aggregate messages

portion that only affects variable i is contained in θ_i. Observe that all layers share the same parameters. The functions $f_{\theta_{ij}}(\cdot)$ are directly derived from $\phi(\boldsymbol{x};\theta)$ using the BP equations, and therefore inherit its parameters. Finally, parameter T is the "temperature" of the message passing, and can be set to $T = 1$ to retrieve the standard sum-product belief propagation or to 0 to recover max-product belief revision. Values in-between interpolate between sum-product and max-product and increase the flexibility of the NN. Refer to the Supplementary Material for the precise equations obtained when applied to the three popular undirected models used in our experiments.

Connection with Pseudo-likelihood

If the query distribution is uniform over all queries with exactly one target variable (the rest being evidence), and if there are no hidden variables, the above cost function reduces to the pseudo-likelihood (PL) of (Besag 1975). Other query distributions can be seen as composite likelihoods. However, unlike PL or composite likelihoods, QT can approximately marginalize hidden variables and does not aim to learn the true parameters of the model, but instead optimizes them for best performance at inference time.

Experiments

Early works in learning undirected PGMs relied on contrastive energies (Hinton 2002; Welling and Sutton 2005). More recent approaches are NVIL (Kuleshov and Ermon 2017) and AdVIL (Li et al. 2019), with the latter being regarded as superior. We will use it as our main benchmark.

We will test QT as an approach for learning and querying in undirected PGMs, comparing it with AdVIL on 3 different types of undirected PGMs (using both discrete and continuous variables) and a total of 10 datasets. These models and datasets are exactly the ones used in AdVIL's paper (Li et al. 2019). We also apply QT to a challenging grid MRF.

Comparison with AdVIL

All models are trained using uniform random query distribution, i.e., in each SGD step, each variable is independently assigned as input or output with 0.5 probability. We report the test normalized cross-entropy (NCE) in bits to measure generalization to new data *and new queries*. We normalize the number of predicted variables, so the NCE is the average CE per-variable. The query distribution at test time is the same as the training one, but the actual queries are with very high probability unseen in training. Experiments are run on a single Tesla V100 GPU.

The QT-NN equations for the models in this section (simply unrolling LBP on each PGM) and code to reproduce our results can be found in the Supplementary Material.

Restricted Boltzmann Machine (RBM) We first test QT on which is arguably the simplest undirected model with hidden variables, the RBM. The log-probability of an RBM[5] is proportional to $\phi(\boldsymbol{v},\boldsymbol{h};\theta) = 2\boldsymbol{h}^\top W\boldsymbol{v}+\boldsymbol{h}^\top(\boldsymbol{c}_H - W\mathbf{1}_V)+\boldsymbol{v}^\top(\boldsymbol{c}_V - W^\top 1_H)$. We use exactly the same datasets and preprocessing used in the AdVIL paper, with the same hidden layer sizes, check Li et al. (2019) for further details. Since all the variables are binary, a trivial uniform undirected model would result in an NCE of 1 bit.

We also include results from persistent contrastive divergence (PCD), which is known to be very competitive for RBM training (Tieleman 2008; Marlin et al. 2010). In fact, AdVIL does not do much better than PCD on this model.

Computing the test NCE for QT only requires to running the trained QT-NN on test data and evaluating its loss function. PCD and AdVIL, however, do not provide any mechanism to solve arbitrary inference queries, so we needed to resort to Gibbs sampling in the learned model, which is much slower. Alternatively, we also tried copying the RBM weights learned by PCD and AdVIL into the QT with $T = 1$ and report those results as PCD-BP and AdVIL-BP.

For AdVIL we use the code provided by the authors. For PCD and QT the validation set is used to choose the learning rate and for early stopping, separately for each dataset. The learning rates considered are $\{0.03, 0.1, 0.3, 1, 3\}$ for PCD and $\{0.001, 0.003, 0.01, 0.03\}$ for QT. For PCD we use `scikit-learn` (Pedregosa et al. 2011). For QT we unfold BP in $N = 10$ layers and use ADAM (Paszke et al. 2019; Kingma and Ba 2014) with minibatches of size 500. The T parameter is learned during training. The results are shown in Table 1. Results average 5 independent runs.

QT produces significantly better results for most datasets (marked in boldface), showing that it has learned to generalize to new probabilistic queries on unseen data.

Deep Boltzmann Machine (DBM) For our second experiment, we use a slightly more sophisticated undirected PGM, a DBM with two hidden layers. The log-probability of a DBM[6] is proportional to $\phi(\boldsymbol{v},\boldsymbol{h};\theta) = 2\boldsymbol{h}_2{}^\top W_{H_2H_1}\boldsymbol{h}_1 + 2\boldsymbol{h}_1{}^\top W_{H_1V}\boldsymbol{v} + \boldsymbol{h}_2{}^\top(\boldsymbol{c}_{H_2} - W_{H_2H_1}\mathbf{1}_{H_1}) + \boldsymbol{v}^\top(\boldsymbol{c}_V - W_{H_1V}^\top 1_{H_1}) + \boldsymbol{h}_1{}^\top(\boldsymbol{c}_{H_1} - W_{H_2H_1}^\top\mathbf{1}_{H_2} - W_{H_1V}\mathbf{1}_V)$. The datasets are the same as for the RBM. The DBM structure (number of units in each hidden layer) for each dataset is identical to that of Li et al., and we reuse exactly the same process and parameters as in the previous experiment. The results are summarized in Table 1.

QT produces the best performance (marked in boldface) on 8 out of the 9 tested datasets. It is interesting to note that, although inference becomes more challenging for DBM than for RBM (as demonstrated by the generally higher NCEs on most datasets for AdVIL-BP and AdVIL-Gibbs), the performance of QT remains essentially unchanged for DBM as compared with RBM. This suggests that QT not only learns to generalize to new probabilistic queries on unseen data, but also remains highly effective for models where inference becomes more challenging.

Gaussian Restricted Boltzmann Machine (GRBM) Finally, we compare QT with AdVIL on learning a GRBM on the Frey faces dataset. The log-probability of a GRBM is proportional to $\phi(\boldsymbol{v},\boldsymbol{h};\theta) = -\frac{1}{2\sigma^2}\|\boldsymbol{v}-\boldsymbol{b}\|^2 + \boldsymbol{c}^\top\boldsymbol{h} +$

[5]This parameterization is in one-to-one correspondence with the standard and results in simpler QT-NN equations as shown in the Supplementary Material. Most readers can ignore this detail.

[6]Previous footnote also applies here.

Method	Model	**Adult**	**Conn4**	**Digits**	**DNA**	**Mushr**	**NIPS**	**OCR**	**RCV1**	**Web**
AdVIL-BP	RBM	0.224	0.248	0.530	0.778	0.192	0.795	0.470	0.475	0.142
AdVIL-Gibbs	RBM	0.229	0.238	0.493	0.782	0.218	0.797	0.471	0.477	0.163
PCD-BP	RBM	0.215	0.285	0.530	0.763	0.159	0.801	0.428	0.457	0.140
PCD-Gibbs	RBM	0.218	0.288	0.516	0.765	0.159	0.804	0.427	0.458	0.144
QT (Ours)	RBM	**0.167**	**0.148**	**0.472**	0.766	**0.124**	**0.787**	**0.377**	**0.452**	**0.133**
AdVIL-BP	DBM	0.242	0.364	0.578	0.805	0.224	0.816	0.490	0.480	0.146
AdVIL-Gibbs	DBM	0.244	0.299	0.529	0.560	0.690	0.934	0.923	0.568	0.195
QT (Ours)	DBM	**0.169**	**0.146**	**0.471**	0.765	**0.124**	**0.787**	**0.379**	**0.453**	**0.133**

Table 1: Comparison of QT-NN, PCD and AdVIL on RBM and DBM. Measurements are NCE, lower is better. QT achieves significantly lower NCE for most datasets compared to the state-of-the-art. Code: https://github.com/vicariousinc/query_training.

$\frac{1}{\sigma}\boldsymbol{h}^\top W\boldsymbol{v}$. The hidden units are binary and the visible units are continuous. We follow Li et al., and fix $\sigma = 1$. The dimensions of $\boldsymbol{h}$ and $\boldsymbol{v}$ are respectively 200 and 560 (corresponding to a 28×20 image).

We train AdVIL using the provided authors' code. For QT we unfold BP in $N = 50$ layers and use ADAM with learning rate 5×10^{-3}. In the GRBM, BP will send continuous messages to the visible units, and we follow the standard practice of expectation propagation (Minka 2001) of characterizing those messages as Gaussians. This results in deterministic and differentiable message updates. For further details, see the Supplementary Material.

Since no quantitative results are provided in (Li et al. 2019), we create our own by using the preexisting train-test split in the dataset of 1572 and 393 images, respectively. Since there is no validation split, for QT we simply stop training after 50 epochs (training performance has plateaued and we see no signs of overfitting). For AdVIL, we train for 100,000 iterations and save a checkpoint every 100 iterations. We observe overfitting, so we decided to give the competing method an additional advantage and we report the results of the checkpoint that provides the best test set performance. Like in the previous experiments, the results for AdVIL are obtained both by using Gibbs sampling and by transferring its learned weights to a QT in which we run $N = 50$ BP iterations. The NCE for each of the models are Advil-Gibbs: 1.545, Advil-BP: 1.542, and QT: $\mathbf{1.503}$. The NCE of a trivial independent Gaussian model using the empirical mean and variance of the training pixels is 1.909. Again, QT produces better results than AdVIL, showing that it can be applied to continuous PGMs.

Testing on significantly different queries. Ideally, the QT-NN should be trained under a query distribution that matches the query statistics at test time. However, even using a conservative, uniform distribution should already be better than optimizing for a single fixed query. In order to assess this we use the GRBM trained on uniform queries and test it on significantly different ones, asking to complete contiguous patches of 5×5 pixels given the rest of the image. The NCE for each of the models are: Advil-Gibbs: 1.525, Advil-BP: 1.530, and QT: $\mathbf{1.493}$. The NCE of the trivial model is 1.889. Qualitatively, we find that QT is able to produce image completions that are almost indistinguishable from the original images (shown in Fig 2a).

This shows the PGM-like flexibility of QT, answering queries on blocks that it was never trained for. Of course, prediction will degrade as the statistics of the test queries depart further from the training ones.

Grid Markov Random Field (GMRF)

We consider the challenging problem of using QT to learn an 8-connected, grid-arranged Markov random field (MRF) *with hidden variables*, as shown in Fig. 2b. Although the models explored so far also had hidden variables, they did not have direct connections—as it is the case now—which make the model more loopy and learning more challenging. Grid MRFs are often used in image processing applications (Li 2009), but the MRF variables are always observed when learning the factor parameters. In our case, the MRF variables are hidden, and they emit the pixel labels through a noisy channel, with multiple hidden states mapping to the same pixel labels.

To the best of our knowledge, QT is the first method that can efficiently learn the full parameterization of an 8-connected grid MRF without direct access to the MRF variables (which are hidden and only visible through a noisy channel). Although irrelevant for our purpose of learning a challenging undirected PGM, the proposed model is a simple incarnation of visual neuroscience principles for foreground–background segmentation. Further details about the model and its neuroscience motivation are presented in the Supplementary Material.

The Border Ownership Dataset, Model, and Task The border ownership dataset is provided with the Supplementary Material and is derived from the MNIST dataset[7] (LeCun, Cortes, and Burges 2010). It is structured as pairs of noisy contour images and `CONTOUR-IN-OUT` labels. Two examples are displayed in Fig. 2c. The contours are missing with probability 0.2, whereas each image incorporates 8 spurious random edges of length 3 pixels. Each image is of size 30×30 pixels. The images have one-to-one correspondence with the MNIST dataset, so 60,000 images are available for training and 10,000 images are used for testing.

The structure of the MRF is shown in Fig. 2b. Each node is a categorical variable with 66 hidden states, 64 corre-

[7]The MNIST dataset can be found at http://yann.lecun.com/exdb/mnist/.

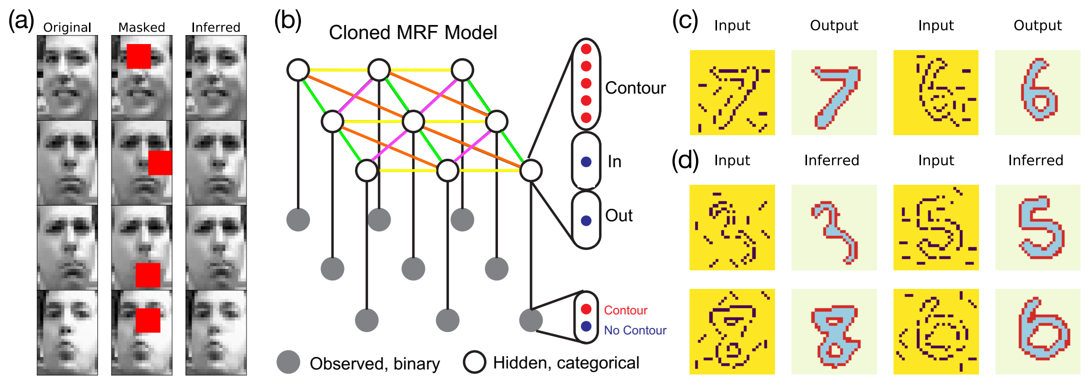

Figure 2: (a) QT can accurately complete masked regions of an image even though it was trained for significantly different queries. (b) An 8-connected cloned Markov random field. Identical factors are shown using the same color. Actual size is 30×30. (c) Two examples of training pairs: a noisy input digit and its corresponding ground truth segmentation (d) Test data: noisy input digits and their inferred segmentation by the QT-NN (which is obtained unrolling the GMRF model).

spond to the label CONTOUR, 1 to IN, and 1 to OUT. The vertical connections are a noisy channel. There are only 4 distinct pairwise factors (different colors in Fig. 2b). The task is to recover the hidden labels from the noisy image. Observe that the incomplete contours and the spurious edges make the task of foreground–background segmentation non-trivial. Second, observe that the labels do *not* provide the hidden states. In particular, which of the 64 clones of CONTOUR is appropriate for each pixel is unknown, and the use of multiple clones is required to properly solve the task, since the potentials are only local pairwise connections and long-range information is needed.

Results The model is trained using QT. The input and output variables in this case are not randomized, but fixed throughout training and testing: the evidence is always the noisy binary image and the target is always the noiseless ternary segmentation. We unroll BP for $N = 15$ layers, use ADAM with a learning rate of 10^{-2} with minibatches of 50 images and run learning for 10 epochs on a single Tesla V100 GPU. The temperature parameter is fixed $T = 1$ throughout learning. The results of segmentation from noisy test data are shown on Fig. 2d for several example digits. Pixels decoded as CONTOUR, IN, OUT are respectively in red, pale blue, and pale yellow. Qualitatively, the recovery looks almost perfect. Quantitatively, Table 2 presents the intersection over union (IoU) between the estimated and real foreground[8], and the NCE of a GMRF trained with QT and with a random baseline. QT achieves a nearly perfect digit recovery.

[8]For the purpose of this metric, we consider foreground those pixels labeled (or estimated) as either IN, CONTOUR. Higher is better.

Method	IOU	NCE
QT (OURS)	**96.97**	**0.0398**
RANDOM	16.39	1.00

Table 2: Results for QT and a random baseline for GMRF on the test border ownership dataset.

We have tried to train the GMRF using other alternatives without luck. In particular, AdVIL requires designing three new encoder networks and one decoder network for this model. Our network designs have failed to produce any meaningful results in a reasonable amount of time.

Discussion and Future Work

Query training is a general approach to parameter learning in PGMs when these need to support flexible querying. It offers the following advantages: (a) simple to implement (just differentiable LBP); (b) applicable to any PGM, even undirected ones with hidden variables; (c) it sidesteps the estimation of the partition function; (d) arbitrary inference is built in (e.g., AdVIL needed separate Gibbs sampling to answer our queries); (e) it adapts learning to inference according to a *query distribution*, ignored in previous works, which can be set to uniform and already provide reasonable generalization capabilities.

Why does QT generalize to new queries? The worry could be that only a small fraction of the exponential number of potential queries is seen during training. The *existence* of a single inference network that works well for many different queries follows from the existence of a single, correct PGM in which LBP can approximate inference reasonably well for arbitrary queries. The *discoverability* of such a network from limited training data is not guaranteed. However, there is hope for it, since the amount of training data required to adjust the model parameters should scale with the number of these, and not with the number of potential queries. Just like training data should come from the same distribution as test data, the training queries must come from the same distribution as the test queries to avoid "query overfitting".

Interesting avenues for exploration are model sampling and other inference mechanisms, such as VI.

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. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. URL https://www.tensorflow.org/. Last visited 2020-05-16.

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

Blei, D. M.; Kucukelbir, A.; and McAuliffe, J. D. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* 112(518): 859–877.

Domke, J. 2011. Parameter learning with truncated message-passing. In *CVPR 2011*, 2937–2943. IEEE.

Hinton, G. E. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation* 14(8): 1771–1800.

Hyvärinen, A. 2006. Consistency of pseudolikelihood estimation of fully visible Boltzmann machines. *Neural Computation* 18(10): 2283–2292.

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

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

Kuleshov, V.; and Ermon, S. 2017. Neural variational inference and learning in undirected graphical models. In *Advances in Neural Information Processing Systems*, 6734–6743.

LeCun, Y.; Cortes, C.; and Burges, C. 2010. MNIST handwritten digit database.

Li, C.; Du, C.; Xu, K.; Welling, M.; Zhu, J.; and Zhang, B. 2019. To relieve your headache of training an MRF, take AdVIL. *arXiv preprint arXiv:1901.08400* .

Li, S. Z. 2009. *Markov random field modeling in image analysis*. Springer Science & Business Media.

Lin, G.; Shen, C.; Reid, I.; and van den Hengel, A. 2015. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, 361–369.

Marlin, B.; Swersky, K.; Chen, B.; and Freitas, N. 2010. Inductive principles for restricted Boltzmann machine learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 509–516.

Minka, T. P. 2001. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, 362–369.

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

Ng, A. Y.; and Jordan, M. I. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, 841–848.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 8024–8035.

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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12: 2825–2830.

Rezende, D. J.; Mohamed, S.; and Wierstra, D. 2014. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082* .

Stoyanov, V.; Ropson, A.; and Eisner, J. 2011. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 725–733.

Sutton, C.; and McCallum, A. 2005. Piecewise training for undirected models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 568–575. AUAI Press.

Sutton, C.; and McCallum, A. 2007. Piecewise pseudolikelihood for efficient training of conditional random fields. In *Proceedings of the 24th international conference on Machine learning*, 863–870. ACM.

Tieleman, T. 2008. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, 1064–1071. ACM.

Titsias, M.; and Lázaro-Gredilla, M. 2014. Doubly stochastic variational Bayes for non-conjugate inference. In *International conference on machine learning*, 1971–1979.

Wainwright, M. J. 2006. Estimating the“Wrong”Graphical Model: Benefits in the Computation-Limited Setting. *Journal of Machine Learning Research* 7(Sep): 1829–1859.

Wainwright, M. J.; Jaakkola, T. S.; and Willsky, A. S. 2003. Tree-reweighted belief propagation algorithms and approximate ML estimation by pseudo-moment matching. In *AISTATS*, volume 3, 3.

Welling, M.; and Sutton, C. A. 2005. Learning in Markov Random Fields with Contrastive Free Energies. In *AISTATS*.

Wiseman, S.; and Kim, Y. 2019. Amortized Bethe Free Energy Minimization for Learning MRFs. In *Advances in Neural Information Processing Systems*, 15546–15557.

Yoon, K.; Liao, R.; Xiong, Y.; Zhang, L.; Fetaya, E.; Urtasun, R.; Zemel, R.; and Pitkow, X. 2019. Inference in probabilistic graphical models by graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 868–875. IEEE.