Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Scalable Training of Markov Logic Networks Using Approximate Counting

Somdeb Sarkhel,[1] Deepak Venugopal,[2] Tuan Anh Pham,[1] Parag Singla,[3] Vibhav Gogate[1]
[1]Department of Computer Science, The University of Texas at Dallas
{*sxs104721, txp112330, vxg112130*}*@utdallas.edu*
[2]Department of Computer Science, The University of Memphis, *dvngopal@memphis.edu*
[3]Indian Institute of Technology Delhi, India, *parags@cse.iitd.ac.in*

Abstract

In this paper, we propose principled weight learning algorithms for Markov logic networks that can easily scale to much larger datasets and application domains than existing algorithms. The main idea in our approach is to use approximate counting techniques to substantially reduce the complexity of the most computation intensive sub-step in weight learning: computing the number of groundings of a first-order formula that evaluate to true given a truth assignment to all the random variables. We derive theoretical bounds on the performance of our new algorithms and demonstrate experimentally that they are orders of magnitude faster and achieve the same accuracy or better than existing approaches.

Markov logic networks (MLNs) (Domingos and Lowd 2009) compactly represent a large family of Markov networks using a set of weighted first-order logic formulas. They are routinely used in a wide variety of application domains including natural language understanding, social networks, and computer vision for modeling relational and uncertain knowledge. A key task that impacts their success in these applications is weight learning: given data and formulas, find a setting of weights such that the likelihood or an approximation to it (e.g., contrastive divergence (Hinton 2002), pseudo log-likelihood (Besag 1975), etc.) is maximized. Unfortunately, existing weight learning approaches (Singla and Domingos 2005; Lowd and Domingos 2007; Huynh and Mooney 2009) are extremely time consuming and are unable to scale "out of the box" to large domains such as information extraction (Poon and Domingos 2007).

The main reason for the poor scalability of existing techniques is the high polynomial complexity of algorithms used for computing the gradient. Specifically, a sub-step in gradient computation is computing the number of groundings of a first-order formula that evaluate to TRUE given a truth assignment to all the ground predicates. *Exact algorithms* for solving this #SG (short for #SatisfiedGroundings) problem have high polynomial complexity. For example, given a formula $\forall d \forall a_1 \forall a_2$ $\texttt{HasAuthor}(d, a_1) \wedge \texttt{HasAuthor}(d, a_2) \Rightarrow \texttt{Coauthor}(a_1, a_2)$ (if a_1 and a_2 are authors of document d, then they are likely to be coauthors), even an advanced, exact algorithm such as the one proposed in (Venugopal, Sarkhel, and Gogate 2015) will incur a complexity of $O(nm^2)$ where n is number of documents and m is the number of authors.

Currently, the most prevalent approach for achieving scalability is to use background knowledge, which is incorporated in the MLN as hard evidence. Hard evidence substantially reduces the complexity of solving the #SG problem, reducing it from $O(s^k)$ to $O(r^k)$ where $r << s$. This is because most ground formulas that contain evidence atoms will evaluate to either TRUE or FALSE, and they can be deleted from consideration. For example, in the entity resolution domain, Singla et al. (Singla and Domingos 2006) achieve scalability by adding hard evidence, setting all atoms that are likely to be false (true) with high probability to FALSE (TRUE). These atoms are identified using the canopy approach of McCallum et al. (McCallum, Nigam, and Ungar 2000) – a cheap, domain-specific heuristic. The main drawbacks of using background knowledge are: (1) the added evidence may be incorrect and thus the approach introduces errors, and (2) the approach will not scale to large domains if the number of evidence atoms is small.

In this paper, we present a principled approach, which can scale seamlessly to large domains, even when background knowledge is not available. Our contributions are three-fold:

- We propose new objective functions for weight learning that approximate well known functions such as likelihood, psuedo likelihood and contrastive divergence. The key idea in these new functions is to use *approximate* instead of exact approaches for solving the #SG problem. We analyze the complexity of our new algorithms and show that they are highly scalable.
- We provide theoretical bounds on how far our solutions are from the solutions obtained using existing exact methods.
- We demonstrate experimentally that our new algorithms are superior to existing weight learning algorithms implemented in Alchemy (Kok et al. 2008) and Tuffy (Niu et al. 2011), two state-of-the-art systems for learning and inference in Markov logic. More importantly, our results show that our new methods do not require background knowledge to scale to large domains.

Background and Preliminaries

Markov Logic Networks (MLNs). MLNs soften the constraints expressed by a first-order knowledge base by attaching a weight to each formula in the knowledge base. Higher

the weight, higher the probability of the clause (we assume the knowledge base to be in clausal form) being satisfied, all other things being equal. MLNs can also be seen as first-order templates for generating large Markov networks. Formally, an MLN is a set of pairs (f_i, θ_i) where f_i is a formula in first-order logic and θ_i is a real number. Given a set of constants, an MLN represents a ground Markov network which has one random variable for each grounding of each predicate and one propositional feature for each grounding of each formula. The weight associated with a feature is the weight attached to the corresponding formula. The ground MLN represents the following probability distribution:

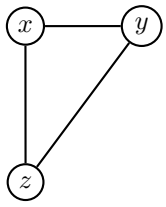
$$P_{\boldsymbol{\theta}}(\omega) = \frac{1}{Z_{\boldsymbol{\theta}}} \exp\left(\sum_i \theta_i N_i(\omega)\right) \quad (1)$$

where $N_i(\omega)$ is the number of groundings of f_i that evaluate to TRUE given a world ω (an assignment to each ground atom) and $Z_{\boldsymbol{\theta}}$ is the normalization constant. We call this the #SG problem. Important inference queries over MLNs are computing the partition function, finding the marginal probability of a variable given evidence (where evidence is an assignment to a subset of variables), and finding the most probable assignment to all variables given evidence, also called the MAP inference problem.

Solving the #SG problem. The main computational bottleneck in popular inference algorithms for MLNs such as Gibbs sampling for marginal inference and MaxWalksat (Kautz, Selman, and Jiang 1997) for MAP inference, is computing $N_i(\omega)$. Until recently, the #SG problem was solved using the following naive method: given a clause f_i and a world ω, generate all possible ground clauses of f_i and count only those that are satisfied in ω. Venugopal et al. (Venugopal, Sarkhel, and Gogate 2015) showed that the problem can be solved efficiently by reducing it to the problem of computing the number of solutions of a constraint satisfaction problem (0/1 Markov network in which all potentials have just two values: 0 and 1). Formally, given a first-order clause f_i and a world ω, the corresponding constraint network $\mathcal{C}_i$ has a variable for each (universally quantified) logical variable in f_i. The domain of each variable in $\mathcal{C}_i$ is the set of constants in the domain of the corresponding logical variable. For each atom $\mathtt{R}(x_1, \ldots, x_u)$ in f_i, we have a constraint ϕ in $\mathcal{C}$ defined as follows: $\phi(\overline{\mathbf{x}}_u) = \omega_{\mathtt{R}(X_1,\ldots,X_u)}$ if $\mathtt{R}$ is negated in f and $\phi(\overline{\mathbf{x}}_u) = 1 - \omega_{\mathtt{R}(X_1,\ldots,X_u)}$ otherwise. Here, $\overline{\mathbf{x}}_u = (x_1 = X_1, \ldots, x_u = X_u)$ denotes an assignment to the variables in the constraint network and $\omega_{\mathtt{R}(X_1,\ldots,X_u)}$ is the projection of the world ω on the ground atom $\mathtt{R}(X_1, \ldots, X_u)$. Thus, $\omega_{\mathtt{R}(X_1,\ldots,X_u)} = 1$ if $\mathtt{R}(X_1, \ldots, X_u)$ is true in ω and 0 otherwise.

Example 1. *Figure 1 shows an encoded constraint network.*

Venugopal et al. (Venugopal, Sarkhel, and Gogate 2015) showed that if $\#\mathcal{C}_i$ denotes the number of solutions of the constraint network $\mathcal{C}_i$ associated with a clause f_i and world ω then: $N_i(\omega) = \prod_{x_j \in V(f_i)} |\Delta(x_j)| - \#\mathcal{C}_i$ where $V(f_i)$ denotes the set of logical variables in f_i and $\Delta(x_j)$ is the set of constants in the domain of x_j. Thus, if a junction tree algorithm is used to compute $\#\mathcal{C}_i$, then its time and space complexity are exponential in treewidth. Since the treewidth can be much smaller than the number of logical variables, Venugopal et al.'s method may be orders of magnitude better than the naive approach, which is exponential in the number of logical variables. Unfortunately, since the number of constants can be quite large (thousands), even treewidth of 3 may be infeasible.



(a) Constraint Net

x	y	val
A	A	0
A	B	1
B	A	1
B	B	0

(b) $\phi(x, y)$

y	z	val
A	A	1
A	B	0
B	A	1
B	B	1

(c) $\phi(y, z)$

x	z	val
A	A	1
A	B	0
B	A	1
B	B	1

(d) $\phi(x, z)$

Figure 1: (a) Constraint network for $f = \forall x, \forall y, \forall z \ \neg\mathtt{R}(x, y) \vee \mathtt{S}(y, z) \vee \mathtt{T}(x, z)$. The domain of each logical variable is $\{A, B\}$; (b), (c) and (d): Constraints $\phi(x, y)$, $\phi(y, z)$ and $\phi(x, z)$ corresponding to the possible world in which the ground atoms $\mathtt{R}(A, B)$, $\mathtt{R}(B, A)$, $\mathtt{S}(A, B)$ and $\mathtt{T}(A, B)$ are true and the rest are false.

Iterative Join Graph Propagation (IJGP): IJGP (Mateescu et al. 2010) is a type of generalized Belief propagation algorithm that can perform approximate solution counting in constraint networks. IJGP runs the same message passing as join or junction tree propagation over the clusters of a join graph rather than a join tree, iteratively. A join graph is a decomposition of functions (constraints) of a constraint network into a graph of clusters that satisfies all the properties required of a valid join tree decomposition except the tree requirement. IJGP is particularly useful since its time as well as space complexity can be controlled using a parameter i called the i-bound. As we increase i, the complexity increases exponentially but so does the accuracy. If i equals treewidth of the network plus one $(w^* + 1)$, IJGP is exact.

Weight Learning. A dataset in MLNs is a relational database, which defines a world ω. We assume that the database contains only TRUE ground atoms and is closed world, namely all ground atoms not in the database are false. Note that we can learn to generalize from even a single world because the clause weights are shared across their groundings. The weights can be learned generatively (or discriminatively) by maximizing the log-likelihood (or conditional log-likelihood) of the database:[1]

$$\ell(\boldsymbol{\theta} : \omega) = \log P_{\boldsymbol{\theta}}(\omega) = \sum_i \theta_i N_i(\omega) - \log Z_{\boldsymbol{\theta}} \quad (2)$$

$$= \sum_i \theta_i N_i(\omega) - \log\left(\sum_{\omega'} \exp\left(\sum_i \theta_i N_i(\omega')\right)\right)$$

Weights that optimize the log-likelihood (a convex function) can be learned using a standard gradient ascent procedure.

[1] We show the equations for generative learning noting that they can be easily derived similarly for discriminative learning

The gradient of the log-likelihood w.r.t θ_i is given by:

$$\frac{\partial}{\partial \theta_i}\ell(\boldsymbol{\theta}:\omega) = N_i(\omega) - \sum_{\omega'} P_{\boldsymbol{\theta}}(\omega')N_i(\omega')$$
$$= N_i(\omega) - \mathbb{E}_{\boldsymbol{\theta}}[N_i(\omega)]$$

where the sum is over all possible databases ω', and $P_{\boldsymbol{\theta}}(\omega')$ is computed using the current weight vector $\boldsymbol{\theta} = (\theta_1, ..., \theta_i, ...)$. The second term in the gradient requires (marginal) inference over the ground Markov network and is $\#\mathcal{P}$-hard in the worst-case. Therefore, in practice, the expectation is approximated using either Markov Chain Monte Carlo (MCMC) sampling or MAP inference. The idea is to replace the expectation by either an average over the worlds sampled using a MCMC procedure or the MAP tuple. The former yields a class of algorithms called contrastive divergence (CD) (Hinton 2002), while the latter yields a class of algorithms called Voted Perceptron (VP) (Collins 2002). An alternative approach is to change the likelihood (objective) function such that the gradient is tractable. An example of such a function is pseudo log-likelihood (PLL) (Besag 1975). For lack of space, we describe PLL in the a longer technical report (Sarkhel et al. 2015).

Existing weight learning algorithms, even those that use alternative criteria such as CD, VP and PLL are impractical because all algorithms require solving the #SG problem in order to compute the gradient. As described earlier, even advanced algorithms (Venugopal, Sarkhel, and Gogate 2015) for solving #SG have high polynomial complexity, bounded by $O(n^{w*+1})$ where w^* is the treewidth of the associated constraint network. Specifically, in CD (and VP), in order to generate a a sample (MAP tuple) via Gibbs sampling (MaxWalksat), one has to visit all ground formulas at least once. In PLL, one has to solve the #SG problem, once for each ground atom. However, in terms of computational complexity, a key advantage of PLL over CD and VP is that the counts remain constant over iterations and therefore they can be precomputed. Thus, PLL will be more scalable than VP and CD if the gradient ascent procedure requires a large number of iterations to converge.

Relaxed Learning Formulation with Approximate Counts

Next, we consider a simple modification of the likelihood function in which instead of computing the exact counts $N_i(\omega)$, we assume that we have access to a procedure that computes approximate counts $M_i(\omega)$ such that the absolute error between the two is bounded by ϵ. Namely, $\forall\omega\ |N_i(\omega) - M_i(\omega)| \leq \epsilon$ where $\epsilon \geq 0$. In this case, our relaxed learning objective is defined as:

$$\underset{\boldsymbol{\theta}}{\text{maximize}}\ \ \hat{\ell}(\boldsymbol{\theta}:\omega) = \sum_i \theta_i M_i(\omega) - \log \hat{Z}_{\boldsymbol{\theta}}$$
$$\text{subject to}\ \ \forall\omega\ |N_i(\omega) - M_i(\omega)| \leq \epsilon$$

where $\hat{Z}_{\boldsymbol{\theta}} = \sum_{\omega'} \exp(\sum_i \theta_i M_i(\omega'))$.

We show that this relaxed learning objective is a reasonable approximation to the original objective in that the difference in the log-likelihood between the parameters learned using the relaxed and original objective is bounded linearly by the sum of parameter values and ϵ. In other words, if the data is generated from an MLN that is fairly smooth in that it does not have large weights, then the distribution learned using the relaxed objective will be fairly close to the one learned using the original objective. We assume that all weights are positive for simplicity of exposition. Note that this is not a limitation of our result since we can easily convert an arbitrary MLN to an equivalent MLN having positive weights by negating each clause having negative weight along with its weight[2]. Formally, if we replace a weighted formula $(f, -\theta)$ by $(\neg f, \theta)$, where $0 < \theta < \infty$ then the distribution represented by the MLN does not change.
Formally, (*proof is included in a longer technical report (Sarkhel et al. 2015)*)

Theorem 1. *Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m)$ denote the parameters (weights) that optimize the function $\ell(\boldsymbol{\theta}:\omega)$ and $\hat{\ell}(\boldsymbol{\theta}:\omega)$ respectively ($\boldsymbol{\alpha}$ is the maximum likelihood estimate of $\boldsymbol{\theta}$) and let ϵ be an upper-bound on the error of the approximate counting algorithm used to estimate $N_i(\omega)$, then*

$$\ell(\boldsymbol{\alpha}:\omega) - \ell(\boldsymbol{\beta}:\omega) \leq 2\epsilon \sum_i (\alpha_i + \beta_i)$$

The power of Theorem 1 is that as the approximate counting algorithm becomes more accurate, the bound specified in Theorem 1 becomes tighter. This allows us to leverage significant advances in approximate counting algorithms to improve the quality of our learning algorithms. Numerous studies have shown that algorithms such as loopy Belief propagation (Murphy, Weiss, and Jordan 1999) and its generalizations (Yedidia, Freeman, and Weiss 2005) often yield highly accurate estimates of the counts and we can use them instead in practice. In our experiments, we use IJGP to compute the approximate counts. It should be noted that IJGP is remarkably effective in practice in estimating the partition function. For instance, it won the 2010 and 2012 UAI approximate inference competitions (Elidan and Globerson 2010; 2011).[3]

Scalable Learning Algorithms using Approximate Counts

In this section, we describe how to scale up the contrastive divergence algorithm using IJGP as the solution counter.

Our approach for scaling up Contrastive Divergence is described in Algorithm 1. The algorithm takes as input a MLN M, an i-bound for IJGP, which determines its complexity, number of samples N for Gibbs sampling, and a dataset ω (world) and outputs the learned weights. The algorithm has

[2]Negating a clause makes it a conjunction. We can easily extend Venugopal et al.'s method for solving #SG to this case.

[3]Another option is to use sampling based algorithms for computing the counts including rejection sampling and advanced algorithms such as `SampleSearch` (Gogate and Dechter 2011), and `Appoxcount` (Gomes et al. 2007). Unfortunately, these algorithms work with binary domains and are very inefficient at processing multi-valued variables.

Algorithm 1 Scalable Contrastive Divergence
(MLN M, i-bound for IJGP, Number of samples N, dataset or world ω, learning rate η, update frequency K)

```
Initialize all weights {θ_j} using a prior.
/* Compute the data counts */
for all formulas f_j do:
    Convert (f_j, ω) to a constraint network C_j
    Compute M_j(ω) using the algorithm IJGP(i)
    and store it in a lookup table.
ω^(0) = ω
/* Gradient Ascent */
Repeat until convergence
    /* Gibbs sampling using Approximate counts */
    for t = 0 to N − 1 do:
        Select a random ground atom S
        Compute Pr(S|ω_{−S}^(t)) using Equation (3)
        Sample a value S = s from Pr(S|ω_{−S}^(t))
        Set ω^(t+1) = (ω_{−S}^(t), S = s)
    /* Weight update */
    for each weight θ_j do
        θ_j = θ_j + η(M_j(ω) − (1/K) Σ_{t=1}^{K} M_j(ω^(t)))
return {θ_j}.
```

three major steps: (1) computing the data counts, (2) the gradient ascent procedure, and (3) Gibbs sampling which is a sub-step of gradient ascent.

To scale up the computation of the data counts, we first convert each formula into a constraint network and then use IJGP(i) instead of an exact algorithm to compute the counts $M_j(\omega)$. We store these counts in a look-up table since they will be used at each iteration of gradient ascent.

The most time consuming sub-step in the gradient ascent procedure is Gibbs sampling (Geman and Geman 1984). In Gibbs sampling, we start with a complete assignment $\omega^{(0)}$ (world) and in each subsequent iteration t, we compute the conditional distribution over a randomly chosen ground atom $\mathtt{S}$, given assignments to all other atoms except $\mathtt{S}$, denoted by $\omega_{-\mathtt{S}}$. To scale up Gibbs sampling, instead of using an exact algorithm to compute the conditional distribution, we use an approximate algorithm. Specifically, the distribution is computed using the following equation:

$$P(\mathtt{S}=s|\omega_{-\mathtt{S}}^{(t)}) \propto \exp\Big(\sum_{f_j \in MB(\mathtt{S})} \theta_j M_j(\omega_{-\mathtt{S}}^{(t)}, \mathtt{S}=s)\Big) \quad (3)$$

where $s \in \{0,1\}$, $MB(\mathtt{S})$ are the formulas in the Markov Blanket of $\mathtt{S}$ (namely the formulas that $\mathtt{S}$ is involved in), θ_j is the (current) weight attached to f_j, $(\omega_{-\mathtt{S}}^{(t)}, \mathtt{S}=s)$ is the composition of the assignments $\omega_{-\mathtt{S}}^{(t)}$ and $\mathtt{S}=s$ and $M_j(\omega_{-\mathtt{S}}^{(t)}, \mathtt{S}=s)$ is computed by running IJGP(i) over the constraint network associated with f_j and $(\omega_{-\mathtt{S}}^{(t)}, \mathtt{S}=s)$.

Note that all CSPs associated with f_j have the same function scopes (set of variables involved in a function is called its scope). Thus, we can use the same cluster graph to run Belief propagation via IJGP over all iterations, indexed by t. Moreover, since $\omega^{(t)}$ and $\omega^{(t+1)}$ differ only by an assignment to just one ground atom, only a few function (potential) values will be different between the CSPs of f_j at iterations t and $t+1$. As a result, we can reuse the messages in iteration t for message passing in iteration $t+1$. In fact, if we use advanced message passing strategy such as the one by Elidan et al. (Elidan, McGraw, and Koller 2006), where messages are ordered by how much each message has changed from one iteration to the next, IJGP is likely to converge quickly. We used this advanced ordering strategy in our experiments and found that IJGP converges within 3 iterations in most cases. Moreover, we do not have to calculate a large number of messages because they are guaranteed not to change between subsequent iterations. For example, if all the incoming messages at a cluster have not changed, the outgoing message will be the same as in the previous iteration.

For lack of space, we skip the description of scaling up VP and PLL, noting that they can be similarly extended using the scalable CD algorithm detailed here as a guide. The time and space complexity of the various weight learning algorithms is described in Table. 1.

Experiments

The fundamental question that we seek to answer through our experiments is, can approximate counting (within learning) improve scalability while learning a highly accurate model? We perform a detailed evaluation to answer the above question and compare the performance of our learning algorithms with those implemented in Alchemy (Kok et al. 2008) and Tuffy (Niu et al. 2011), two state-of-the-art MLN learning systems. We conducted our experiments on three datasets publicly available on the Alchemy website. We used three datasets: WebKB, Entity Resolution (ER) and Protein Interaction (Protein). For sanity check, we added another dataset called *Smoker*, that we generated randomly for the Friends and Smokers MLN in Alchemy. Table 2 shows the details of our datasets.

Methodology

We implemented three different algorithms: Contrastive Divergence (CD), Voted Perceptron (VP) and Pseudo Log Likelihood based learning (PLL). Each algorithm performed approximate counting using IJGP. In each algorithm, we implemented the diagonal newton method (Lowd and Domingos 2007) to adapt the learning rate dynamically. For all these algorithms, on all four datasets, we performed five-fold cross-validation. In each fold's test set, we measured the conditional log-likelihood (CLL). The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The CLL of a query predicate is the average log-probability over all ground atoms of the query predicate given the evidence. It should be noted that computing the CLL is also intractable. Therefore, we use Gibbs sampling to estimate the probability of each ground atom corresponding to the query predicate. We collected $100,000$ Gibbs samples to estimate each probability.

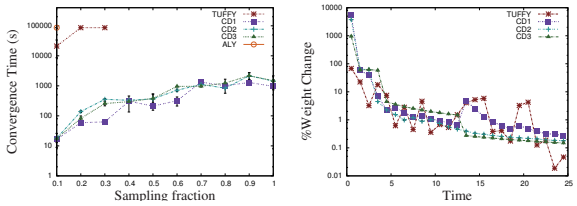
For approximate counting within our learning algorithms, we used the IJGP algorithm. Specifically, for any formula f and world ω, when to estimate the number of satisfied groundings of f in ω, we run IJGP on the CSP encoding for (f, ω) for up to 10 iteration.

Algorithm	Space Complexity	Time Complexity
Contrastive Divergence	$O(F \exp(\min(i, w^*)))$	$O(MNFT \exp(\min(i, w^* + 1)))$
Voted Perceptron	$O(F \exp(\min(i, w^*)))$	$O(MNFT \exp(\min(i, w^* + 1)))$
Pseudo Likelihood	$O(F \exp(\min(i, w^*)) + AF)$	$O(AFT \exp(\min(i, w^* + 1)) + MAF)$

Table 1: Time and Space Complexities for different learning algorithms (showing their scalability potential). M is the number of gradient ascent steps, N is the number of samples (contrastive divergence via Gibbs sampling) or MaxWalkSAT flips (voted perceptron), F is the number of formulas in the MLN, A is the total ground atoms in the MLN and T be the number of IJGP iterations. i is a parameter that controls the maximum cluster size for IJGP and w^* is the maximum treewidth of the encoded CSPs.

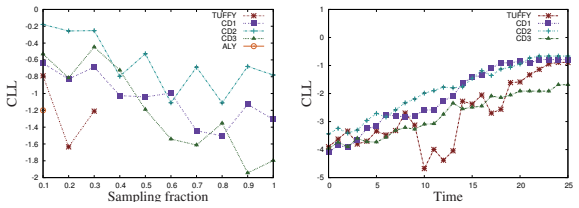
Dataset	#Clauses	#Atoms	#Parameters	#Evidence
Smoker	125K	63K	5	7.5K
WebKB	892 million	20 million	64	227.5K
Protein	408 million	3.3 million	211	25.8K
ER	1.7 trillion	5.5 million	15	56K

Table 2: Dataset sizes.#Evidence is number of true evidence



(a) (b)

Figure 2: Convergence plots. (a) shows a plot of convergence time vs data size for Protein; (b) shows a plot of change in weights vs time (in hours) for WebKB. ALY corresponds to Alchemy and CDw corresponds to contrastive divergence with i-bound w



(a) (b)

Figure 3: Accuracy plots. (a) shows a plot of CLL vs data size for Protein; (b) shows a plot of CLL vs training time (in hours) for WebKB. ALY corresponds to Alchemy and CDw corresponds to contrastive divergence with i-bound w

Results

Table 3 illustrates our results on the different benchmarks. For each benchmark, we divide the data into 5 sets and use cross-validation to obtain the CLL value for each fold. For each algorithm, we also put a time bound of 96 hours (4 days) and a memory bound of 8 GB. Table 3 shows the average CLL value obtained from the 5 folds along with the standard deviation. We compare the results of our 3 learning algorithms with 3 other algorithms implemented in Alchemy: VP, CD and a variant of CD using MC-SAT instead of Gibbs sampling, as well as the MC-SAT based learning algorithm implemented in Tuffy.

As seen in Table 3, we could only run Alchemy on the Smoker dataset, as it ran out of memory while creating the ground Markov network from the data for all other datasets. For the same reason we could only run Tuffy on WebKB and Smoker only (since our methods do not require grounding the MLN, we could scale up to larger MLNs). Among CD, VP and PLL, the results were fairly similar for WebKB, Protein and the ER datasets. CD was marginally better (larger CLL) than VP and PLL on these datasets. For the smokers problem, CD was much better than VP or PLL. The Alchemy learning algorithms performed similar to VP and PLL for the Smoker dataset. However, we could not obtain the results for Alchemy-CD since the weights it learned were infinite for the Smoker dataset. In the case of Tuffy, our CD algorithm performed better than Tuffy (which uses MC-SAT as its inference subroutine) in both WebKB as well as the Smoker datasets. VP performed marginally worse than Tuffy for WebKB and PLL performed marginally better than Tuffy for WebKB. For the Smoker dataset, Tuffy performed better than VP and PLL.

As mentioned in earlier sections, we can control the space/time complexity in IJGP using a parameter (i-bound) that specifies the maximum cluster-size of the join graph that IJGP operates on. The complexity of learning increases as we increase the value of i-bound. Table 4 illustrates the results of using different i-bound values in IJGP with the CD algorithm. For larger i-bounds, the counts are exact (indicated by a $*$ in Table 4). We can see that for the WebKB and ER datasets, for i-bound > 1, the weights did not converge. For the Protein dataset, the weights converged for all values of i-bound. For i-bound $= 1$, the algorithm converged very quickly on all three datasets. Furthermore, the CLL scores did not vary by much as we increase the i-bound. Thus, the difference in results when using approximate counts is almost as good as the results obtained using exact counts. Empirically, this suggests that we only need a "reasonably good estimate" of the partition function from each encoded constraint network to move along the correct gradient direction.

We further illustrate the scalability and accuracy of our approach through a set of 4 experiments shown in figures 2 and 3. Owing to space constraints, we only show results for the Protein and WebKB datasets with the CD algorithm.

In Fig. 2 (a), we vary the size of the training data by subsampling the original datasets according to the parameter k

System	WebKB	Protein	ER	Smoker
Alchemy-VP	X	X	X	-1.21 ± 0.03
Alchemy-MCSAT	X	X	X	-1.21 ± 0.02
Alchemy-CD	X	X	X	$*$
Tuffy-MCSAT	-0.89 ± 0.05	X	X	-0.698 ± 0.04
CD	$-0.66 \pm 0.004(i=2)$	$-0.779 \pm 0.0002(i=2)$	$-0.694 \pm 1 \times 10^{-6}(i=2)$	$-0.695 \pm 0.01(i=2)$
VP	$-0.91 \pm 0.001(i=3)$	$-0.78 \pm 0.001(i=2)$	$-0.693 \pm 2 \times 10^{-6}(i=3)$	$-1.16 \pm 0.18(i=3)$
PLL	$-0.72 \pm 0.001(i=3)$	$-0.74 \pm 0.0004(i=3)$	$-0.693 \pm 2 \times 10^{-5}(i=3)$	$-1.61 \pm 0.08(i=3)$

Table 3: Results on benchmark datasets. The average and standard deviation of the CLL score for five-fold cross validation is shown. Alchemy-VP is the Alchemy version of VP, Alchemy-CD is the Alchemy version of CD using Gibbs sampling and Alchemy-MCSAT is a version of contrastive divergence using MC-SAT, Tuffy-MCSAT is the Tuffy version of contrastive divergence using MC-SAT. X means that we ran out of memory when running the algorithm. $*$ indicates that Alchemy-CD gave us an error during weight learning.$(i = *)$ reports the i-bound that gave the best result for our respective algorithms.

Benchmark	i-bound	CLL	Time (seconds)
ER	1	-0.6943	653
	2	-0.693	X
	3$*$	-0.693	X
WebKB	1	-0.788	92,327
	2$*$	-0.664	X
	3$*$	-1.686	X
Protein	1	-0.9948	178.2
	2$*$	-0.7797	293
	3$*$	-0.7795	260

Table 4: Effect of IJGP i-bound on convergence. Convergence times for the CD algorithm when using different IJGP bounds for approximate counting. Larger bounds imply larger complexity. X indicates that we could not converge. $*$ indicates that counting is exact as that i-bound is larger than the treewidth of the formulas.

$(0 \leq k \leq 1)$ and plot the convergence times for different data-sizes. k, known as *Sampling fraction*, is the ratio of sample size to population size. As Fig. 2 (a) shows, Alchemy cannot scale up beyond $k = 0.1$, Tuffy on the other hand is slightly more scalable and stops after $k = 0.3$. In contrast our approach can easily scale up to all values of k. Even more importantly, it converges much faster than Tuffy and Alchemy as seen for small k. Fig. 2 (b) shows similar results and the weights learned by our approach converges rapidly while the variance in the weights learned by Tuffy is much larger. Further, as shown in our results, even for lower i-bound values (which means increased scalability), the convergence times during learning are fairly comparable.

Finally, Fig. 3 illustrates the accuracy of our approach as compared to Alchemy and Tuffy. Fig 3 (a) shows the CLL for different sizes of training data (controlled using the parameter k). Similar to the previous result, Alchemy cannot process training data beyond $k = 0.1$ while Tuffy fails after $k = 0.3$. In contrast our approach is much more scalable. At the same time, the accuracy of our approach is higher as compared to Tuffy/Alchemy. Further, as shown in our results, using approximate counting (i-bound is less than treewidth) the learning accuracy is quite comparable to the accuracy obtained using exact counting (i-bound equal to treewidth). This proves, apart from our theoretical result, that the accuracy of our approach is not diminished as a result of our approximate counting. Similar results are observed in Fig. 3 (b) where we illustrate CLL vs training time.

Discussion

The #SG problem can be modeled quite easily as computing an aggregate query in databases. Therefore, query optimization techniques (Vardi 1982; Niu et al. 2011) can be leveraged for solving #SG. However, these methods are *exact* and thus have the same drawbacks as the approach in (Venugopal, Sarkhel, and Gogate 2015). In contrast, here we proposed an approximate solution that is far more scalable. Combining our approximation with database query optimization is an interesting direction for future work.

Recently, *lifted inference* methods have been proposed to scale up weight learning in MLNs (Ahmadi et al. 2013; Van den Broeck, Meert, and Davis 2013). However, a major disadvantage with these approaches is that they fail to scale up when the MLN does not have exploitable symmetries. For instance, in the case of discriminative learning, it is well-known that evidence breaks symmetries in the MLN (Van den Broeck and Darwiche 2013) and thus, lifted inference methods are ineffective in such cases. As a result, all lifted approaches to weight learning learn weights generatively. Our approach on the other hand, is applicable for both generative and discriminative learning (as well as alternative learning objectives such as PLL). Further, since our approach complements lifted approaches (#SG is required in lifted inference), in future, we aim to combine our approach with lifted methods (especially approximate lifting methods such as, (Van den Broeck and Darwiche 2013; Sarkhel, Singla, and Gogate 2015; Venugopal and Gogate 2014)) to further enhance scalability.

Conclusion

Weight learning in MLNs is a challenging task. Even algorithms that optimize alternate, approximate learning objectives such as CD, VP and PLL are unable to scale up to large datasets and complex MLNs. To address this scalability issue, practitioners often have to employ background knowledge and ad-hoc solutions. In this paper, we presented a principled approach using approximate counting for scaling up weight learning to much larger datasets than is possible today. We empirically and theoretically showed the power of our approach and that it is not only orders of magnitude more scalable but also much more accurate when compared

to state-of-the-art weight learning algorithms implemented in MLN software packages such as Alchemy and Tuffy.

Acknowledgments

This research was funded in part by the DARPA Probabilistic Programming for Advanced Machine Learning Program under AFRL prime contract number FA8750-14-C-0005 and by the NSF award 1528037. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, AFRL, NSF or the US government.

References

Ahmadi, B.; Kersting, K.; Mladenov, M.; and Natarajan, S. 2013. Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. *Machine Learning* 92(1):91–132.

Besag, J. 1975. Statistical Analysis of Non-Lattice Data. *The Statistician* 24:179–195.

Collins, M. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 1–8. Philadelphia, PA: ACL.

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool.

Elidan, G., and Globerson, A. 2010. The 2010 UAI Approximate Inference Challenge. Available online at: http://www.cs.huji.ac.il/project/UAI10/index.php.

Elidan, G., and Globerson, A. 2011. The Probabilistic Inference Challenge (PIC 2011). Available online at: http://www.cs.huji.ac.il/project/PASCAL.

Elidan, G.; McGraw, I.; and Koller, D. 2006. Residual Belief Propagation: Informed Scheduling for Asynchronous Message Passing. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, 165–173. Arlington, Virginia: AUAI Press.

Geman, S., and Geman, D. 1984. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6:721–741.

Gogate, V., and Dechter, R. 2011. SampleSearch: Importance Sampling in Presence of Determinism. *Artificial Intelligence*

Gomes, C. P.; Hoffmann, J.; Sabharwal, A.; and Selman, B. 2007. From Sampling to Model Counting. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2293–2299.

Hinton, G. E. 2002. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation* 14(8):1771–1800.

Huynh, T. N., and Mooney, R. J. 2009. Max-Margin Weight Learning for Markov Logic Networks. In *Machine Learning and Knowledge Discovery in Databases*. Springer. 564–579.

Kautz, H.; Selman, B.; and Jiang, Y. 1997. A General Stochastic Approach to Solving Problems with Hard and Soft Constraints. In *The Satisfiability Problem: Theory and Applications*. American Mathematical Society.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D.; Wang, J.; and Domingos, P. 2008. The Alchemy System for Statistical Relational AI. Technical Report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://alchemy.cs.washington.edu.

Lowd, D., and Domingos, P. 2007. Efficient Weight Learning for Markov Logic Networks. In *Principles of Knowledge Discovery in Databases*, 200–211.

Mateescu, R.; Kask, K.; Gogate, V.; and Dechter, R. 2010. Iterative Join Graph Propagation algorithms. *Journal of Artificial Intelligence Research* 37:279–328.

McCallum, A.; Nigam, K.; and Ungar, L. 2000. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 169–178.

Murphy, K. P.; Weiss, Y.; and Jordan, M. I. 1999. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 467–475.

Niu, F.; Ré, C.; Doan, A.; and Shavlik, J. W. 2011. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. *PVLDB* 4(6):373–384.

Poon, H., and Domingos, P. 2007. Joint Inference in Information Extraction. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, 913–918. Vancouver, Canada: AAAI Press.

Sarkhel, S.; Singla, P.; and Gogate, V. 2015. Fast Lifted MAP Inference via Partitioning. In *Advances in Neural Information Processing Systems*.

Sarkhel, S.; Venugopal, D.; Pham, T; Singla, P and Gogate, V. 2015. Scalable Training of Markov Logic Networks using Approximate Counting. Technical Report, Department of Computer Science, The University of Texas at Dallas, Richardson, TX.

Singla, P., and Domingos, P. 2005. Discriminative Training of Markov Logic Networks. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*

Singla, P., and Domingos, P. 2006. Entity Resolution with Markov Logic. In *Proceedings of the Sixth IEEE International Conference on Data Mining*

Van den Broeck, G., and Darwiche, A. 2013. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *Advances in Neural Information Processing Systems 26*, 2868–2876.

Van den Broeck, G.; Meert, W.; and Davis, J. 2013. Lifted Generative Parameter Learning. In *AAAI 2013 Workshop on Statistical Relational AI*.

Vardi, M. Y. 1982. The Complexity of Relational Query Languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 137–146.

Venugopal, D. and Gogate, V. 2014. Evidence-Based Clustering for Scalable Inference in Markov Logic.. *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD* :258–273.

Venugopal, D.; Sarkhel, S.; and Gogate, V. 2015. Just Count the Satisfied Groundings: Scalable Local-Search and Sampling Based Inference in MLNs. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2005. Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms. *IEEE Transactions on Information Theory*.