Lifted Parameter Learning in Relational Models

Babak Ahmadi[1] BABAK.AHMADI@IAIS.FRAUNHOFER.DE
Kristian Kersting[1,2,3] KRISTIAN.KERSTING@IAIS.FRAUNHOFER.DE
Sriraam Natarajan[3] SNATARAJ@WAKEHEALTH.EDU
[1]Fraunhofer IAIS, Knowledge Discovery Department, Sankt Augustin, Germany
[2]University of Bonn, Institute of Geodesy and Geoinformation, Bonn, Germany
[3]Wake Forest University, School of Medicine, Winston-Salem, USA

Abstract

Lifted inference approaches have rendered large, previously intractable probabilistic inference problems quickly solvable by employing symmetries to handle whole sets of indistinguishable random variables. Still, in many if not most situations training relational models will not benefit from lifting: symmetries within models easily break since variables become correlated by virtue of depending asymmetrically on evidence. An appealing idea for such situations is to train and recombine local models. This breaks long-range dependencies and allows to exploit lifting within and across the local training tasks. Moreover, it naturally paves the way for online training for relational models. Specifically, we develop the first lifted stochastic gradient optimization method with gain vector adaptation, processing each lifted piece one after the other.

1. Introduction

Statistical relational models provide powerful formalisms to compactly represent complex real-world domains. Unfortunately, computing the exact gradient in such models and hence learning the parameters with exact maximum-likelihood training using current optimization methods like conjugate gradient and limited-memory BFGS is often not feasible as it requires computing marginal distributions of the entire underyling graphical model. Since inference is posing major computational challenges one has to resort to approximate learning by local training using a pseudolikelihood approach or a global propagation algorithm like BP for approximate partial derivatives (Lee et al., 2007). Recently, there have been some advances in learning SRL models, especially for Markov Logic Networks (MLN) (Kok & Domingos, 2009; 2010; Khot et al., 2011; Richardson & Domingos, 2006). Though all these methods exhibit good empirical performance, they apply the closed-world assumption, i.e., whatever is unobserved in the world is considered to be false. To deal with missing information, algorithms based on classical EM have been developed for ProbLog, CP-logic, PRISM, probabilistic relational models, Bayesian logic programs as well as gradient-based approaches for relational models with complex combining. All these approaches, however, assume a *batch* learning setting; only Huynh and Mooney (2011) have recently studied online training of MLNs. Here, training was posed as an online max margin optimization problem and a gradient for the dual was derived and solved using incremental-dual-ascent algorithms. One attractive avenue to scale relational learning is based on lifted message-passing approaches (Singla & Domingos, 2008; Kersting et al., 2009). They have rendered large, previously intractable probabilistic inference problems quickly solvable by employing symmetries to handle whole sets of indistinguishable random variables. Still, in many if not most situations training relational models will not benefit from lifting: *Symmetries within a model easily break since variables become correlated by virtue of depending asymmetrically on evidence.* Thus, lifting produces new models that are often not far from propositionalized, therefore canceling the benefits of lifting for training. This might explain why this is the first work tackling lifted parameter learning. Moreover, in relational learning we often face a single mega-example. Consequently, many if not all standard statistical learning methods do not naturally carry over to the relational case. Consider e.g. stochastic gradient methods that update the weight vector in an online setting. They often scale sub-linearly with the amount of training data, making

This paper is a shortened version of (Ahmadi et al., 2012) them very attractive for statistial relational learning. Empirically, they are even found often to be more resilient to errors made when approximating the gradient. Unfortunately, *stochastic gradients coincide with batch gradients in the relational case since there is only a single mega-example.* In this paper, we demonstrate how to overcome both limitations. To do so, we shatter the full model into pieces. In each iteration, we train the pieces independently and re-combine the learned parameters from each piece Breaking long-range dependencies allows one — as we will show — to exploit lifting across the local training tasks. It also paves the way for online training of relational models since we can treat (mini-batches of) pieces as training examples and process one piece after the other. Based on this insight, we develop our main algorithmic contribution: the first lifted online training approach for relational models using stochastic gradient optimization method with gain vector adaptation based on natural gradients. As our experimental evaluation demonstrates, it already results in considerable efficiency gains, simply because unlike batch training it starts optimizing long before having seen the entire mega-example even once. However, we can do considerably better. The way we shatter the full model into pieces greatly effects the learning quality. Important influences between variables might get broken. To overcome this, we randomly grow relational piece patterns that form trees. Our experimental results show that *tree pieces* can balance well lifting and quality of the online training.

We proceed as follows. We recap Markov logic networks, the probabilistic relational framework we focus on for illustration purpose. Then, we develop the stochastic relational gradient framework. Before concluding, we present our experimental evaluation.

2. Lifted Online Training

We develop our lifted online training method within the framework of Markov logic network (MLN) (Richardson & Domingos, 2006) but it naturally carries over to other relational frameworks. An MLN is defined by a set of first-order formulas F_i with associated weights w_i, $i \in \{1, \ldots, k\}$. Together with a set of constants $C = \{C_1, C_2, \ldots, C_n\}$ it can be grounded to define a markov network. The joint probability distribution of an MLN is given by $P(X = x) = Z^{-1} \exp\left(\sum_i^F \theta_i n_i(x)\right)$ where for a given possible world x, i.e. an assignment of all variables X, $n_i(x)$ is the number of times the ith formula is evaluated *true* and Z is a normalization constant.

The standard parameter learning task for Markov networks can be formulated as follows. Given a set of training instances $D = \{D_1, D_2, \ldots D_n\}$ each constisting of an assignment to the variables in X the goal is to output a parameter vector θ specifying a weight for each $f_k \in F$. Typically, however, only a single mega-example — a large set of inter-connected facts — is given, denoted as E. To train the model, we can seek to maximize the log-likelihood function $\log P(D \mid \theta)$ given by $\ell(\theta, D) = \frac{1}{n} \sum_D \log P_\theta(X = x_{D_n})$, typically by a gradient-descent approach.The gradient of the likelihood function is given by:

$$\partial \ell(\theta, D)/\partial \theta_k = f_k(D) - M \mathbf{E}_{\mathbf{x} \sim P_\theta}[f_k(\mathbf{x})] \qquad (1)$$

This gradient expression has a particularly intuitive form: the gradient attempts to make the feature counts in the empirical data equal to their expected counts relative to the learned model. Note that, to compute the expected feature counts, we must perform inference relative to the current model at every step of the gradient process. Consequently, there is a close interaction between the training approach and the inference method employed for training.

Lifted Belief propagation (LBP) approaches (Singla & Domingos, 2008; Kersting et al., 2009) have recently drawn a lot of attention as they render large previously intractable models quickly solvable by exploiting symmetries. Therefore, they automatically group nodes and potentials of the graphical model into supernodes and superpotentials if they have identical computation trees (i.e., the tree-structured unrolling of the graphical model computations rooted at the nodes). LBP then runs a modified BP on this lifted (clustered) network simulating BP on the propositional network obtaining the same results.

In the fully observed case we can simply count how often a clause is true. Unfortunately, in many real-world domains, the mega-example available is incomplete, i.e., the truth values of some ground atoms may not be observed. For instance in medical domains, a patient rarley gets all of the possible tests. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. As a numerical optimization problem, it typically involves nonlinear, iterative optimization and multiple calls to a relational inference engine as subroutine.

When learning parameters of a given model for a given set of observations the presence of non symmetrical evidence on the variables mostly destroys the symmetries making lifted approaches virtually of no use and most lifted approaches basically fall back to the ground variants. Thus we need to seek a way to make the learning task tractable. An appealing idea for efficiently training large models is to divide the model into pieces that are trained independently and to exploit symmetries across multiple pieces for lifting.

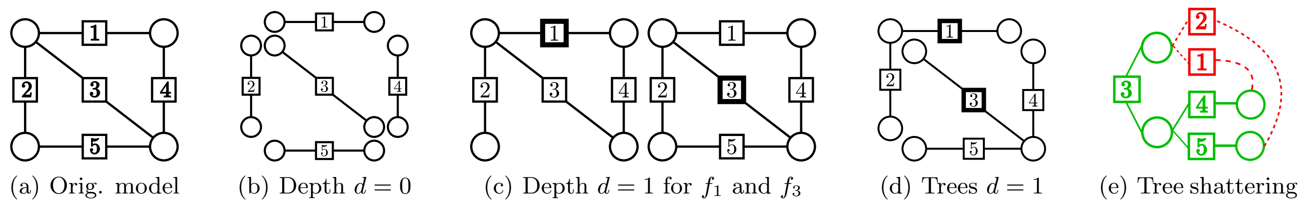

Figure 1. Schematic factor-graph depiction of the difference between likelihood **(a)**, standard piecewise **(b,c)** and treewise training **(d)** (Circles denote variables, boxes denote factors). **(e)** Tree shattering for factor f_3 from the original model.

Piecewise Shattering: In piecewise training, we decompose the mega-example and its corresponding factor graph into tractable, not necessarily disjoint subgraphs (or pieces) $\mathcal{P} = \{p_1, \ldots, p_k\}$ that are trained independently (Sutton & McCallum, 2009). Intuitively, the pieces turn the single mega-example into a set of many training examples and hence pave the way for online training. In many applications, the local information in each factor is already enough to do well at predicting the outputs. The parameters learned locally are then used to perform global inference on the whole model.

More formally, at training time, each piece from $\mathcal{P} = \{p_1, \ldots, p_k\}$ has a local likelihood as if it were a seperate graph, i.e., training example and the global likelihood is estimated by the sum of its pieces: $\ell(\theta, D) = \sum_{p_i \in \mathcal{P}} \ell(\theta|_{p_i}, D|_{p_i})$. Here $\theta|_{p_i}$ denotes the parameter vector containing only the parameters appearing in piece p_i and $D|_{p_i}$ the evidence for variables appearing in the current piece p_i. The standard piecewise decomposition breaks the model into a seperate piece for each factor, intuitively discarding dependencies of the model parameters Although the piecewise model helps to significantly reduce the cost of training the way we shatter the full model into pieces greatly effects the learning and lifting quality. Strong influences between variables might get broken. Consequently, we next propose a shattering approach that aims at keeping strong influence but still features lifting.

Relational Tree Shattering: Assume that the mega-example has been turned into a single factor graph for performing inference, cf. Fig. 1**(a)**. Starting from each factor, we extract networks of depth d rooted in this factor. A local network of depth $d = 0$ thus corresponds to the standard piecewise model as shown in Fig. 1**(b)**, i.e. each factor is isolated in a seperate piece. Networks of depth $d = 1$ contain the factor in which it is rooted and all of its direct neighbors, Fig. 1**(c)**. Thus when we perform inference in such local models using say belief propagation (BP) the messages in the root factor of such a tree resemble the BP messages in the global model up to the d-th iteration. Longer range dependencies are neglected. A small value for d keeps the pieces small and makes inference and hence training more efficient, while a large d is more accurate. However, it has a major weakness: pieces of densely connected networks may contain considerably large subnetworks, rendering the standard piecewise learning procedure useless.

To overcome this, we now present a shattering approach that randomly grows piece patterns forming trees. A tree of factors can then be generalized into a tree pattern, i.e., conjunctions of relational "clauses" by variablizing their arguments. For every clause of the MLN we thus form a tree by performing a random walk rooted in one ground instance of that clause. This process can be viewed as a form of relational pathfinding (Richards & Mooney, 1992). The relational treefinding is summarized in Alg. 1. The actual parameter learning (omitted here due to space restrictions) is interleaved with the relational treefinding such that we can learn paramters before having seen the whole data set. For a given set of Clauses C and a mega example E the algorithm starts off by constructing a tree pattern for each clause C_i **(lines 1-12)**. Therefore, it first selects a random ground instance f_i **(lines 3)** from where it grows the tree. Then it performs a breadth-first traversal of the factors neighborhood and samples uniformly whether they are added to the tree or not **(lines 7)**. If the sample p is larger than $t^{|P_i|}$, where $t \in [0, 1]$ is a discount threshold and $|P_i|$ the size of the current tree, the factor and its whole branch are discarded and skipped in the breadth-first traversal, otherwise it is added to the current tree **(lines 8-11)**. A small t basically keeps the size of the tree small while larger values for t allow for more factors being included in the tree. The procedure is carried out to a depth of at most d, and then stops growing the tree. This is then generalized into a piece-pattern by variablizing its arguments **(line 12)**. All pieces are now constructed based on these piece patterns. For f_i we apply the pattern P_k of clause C_k which generated the factor **(lines 13-15)**. These *tree-based pieces* can balance efficiency and quality of the parameter estimation well. Fig. 1(e) shows the tree rooted in the factor f_3 where green colors show that the factors have been included in the piece while all red factors have been discarded. The neighborhood of

Algorithm 1: RelTreeFinding: Relational Treefinding

```
Input: Set of clauses $\mathbf{C}$, a mega example $\mathbf{E}$, depth $\mathbf{d}$,
       and discount $t \in [0,1]$
Output: Set of tree pieces $\mathbf{T}$
// Tree-Pattern Finding
1  Initialize the dictionary of tree patterns to be empty, i.e.,
   $P = \emptyset$ ;
2  for each clause $C_i \in C$ do
3      Select a random ground instance $F_i$ of $C_i$ in $E$;
4      Initialize tree pattern for $F_i$, i.e., $P_i = \{F_i\}$ ;
       // perform random walk in a breath-first
          manner starting in $F_i$
5      for $F_k$ = BFS.next() do
6          if current_depth $> d$ then break;
7          sample $p$ uniformly from $[0,1]$ ;
8          if $p > t^{|P_i|}$ or $F_k$ would induce a cycle then
9              skip branch rooted in $F_k$ in $BFS$ ;
10         else
11             add $F_k$ to $P_i$;
12     Variablilize $P_i$ and add it to dictonary $P$;
// Construct tree-based pieces using the
   relational tree patterns
13 for each $F_j \in \mathbf{E}$ do
14     Find $P_k \in P$ matching $F_j$, i.e., the tree pattern
       rooted in the clause $C_k$ corresponding to factor $F_j$;
15     Unify $P_k$ with $F_j$ to obtain piece $T_j$ and add $T_j$ to $T$ ;
16 return $T$;
```

factor f_3 is traversed in a breadth-first manner, i.e., first its direct neighbors in random order. Assume we have reached factor f_4 first. We uniformly sample a $p \in [0,1]$. If it is small enough, e.g. $p = 0.3 < 0.9^1$, f_4 is added to the tree. We proceed until we have reached the maximum depth or we cannot add any more edges without including cycles. In this way we can include longer range dependencies in our pieces without sacrificing efficiency. The connectivity of a piece and thereby its size can be controlled via the discount t. By forming tree patterns and applying them to all factors we ensure that we have a potentially high amount of lifting: *Since we have decomposed the model into smaller pieces, the influence of the evidence is limited to a shorter range and hence features lifting the local models.* Moreover, we get an upper bound on the log partition function $A(\Theta)$, that is,

$$A(\Theta) \leq \sum\nolimits_t A(\Theta|_t) \leq \sum\nolimits_r A(\Theta|_r) \ ,$$

where $\Theta|_t$ is the vector Θ with zeros in all entire that do not correspond to the tree t. With $\Theta|_r$ we denote the same vector for pieces of depth 0. The proof is ommitted due to space restrictions. Now, we show how to turn this upper bound into a lifted online training for relational models.

Lifted Stochastic Meta-Descent Stochastic gradient descent algorithms update the weight vector in an online setting. We essentially assume that the pieces are given one at a time. The algorithms examine the current piece and then update the parameter vector accordingly. They often scale sub-linearly with the amount of training data, making them very attractive for large training data as targeted by statistial relational learning. To reduce variance, we may form *mini-batches* consisting of several pieces on which we learn the parameters locally. In contrast to the propositional case, however, mini-batches have another important advantage: we can now make use of the symmetries within and *across* pieces for lifting. More formally, the gradient in (1) is approximated by $\sum_i \frac{1}{\#_i} \frac{\partial \ell(\theta, D_i)}{\partial \theta_k}$, where the mega-example D is partitioned into pieces respectively mini-batches of pieces D_i. Here $\#_i$ denotes a per-clause normalization that counts how often each clause appears in mini-batch D_i. This is a major difference to the propositional case and avoids "double counting" parameters. For a single piece we count how often a ground instance of each clause appears in the piece D_i. If D_i consists of more than one piece we add the count vector of all pieces.

Since the gradient involves inference per batch only, inference is again feasible and more importantly liftable as we will show in the experimental section. Consequently, we can scale to problem instances traditional relational methods can not easily handle. However, the asymptotic convergence of first-order stochastic gradients to the optimum can often be painfully slow if e.g. the step-size is too small. One is tempted to just employ standard advanced gradient techniques such as L-BFGS. Unfortunately most advanced gradient methods do not tolerate the sampling noise inherent in stochastic approximation: it collapses conjugate search directions (Schraudolph & Graepel, 2003) and confuses the line searches that both conjugate gradient and quasi-Newton methods depend upon. Gain adaptation methods like Stochastic Meta-Descent (SMD) overcome these limitations by using second-order information to adapt a per-parameter step size (Vishwanathan et al., 2006). However, while SMD is very efficient in Euclidian spaces, Amari (Amari, 1998) showed that the parameter space is actually a Riemannian space of the metric C, the covariance of the gradients. Consequently, the ordinary gradient does not give the steepest direction of the target function. The steepest direction is instead given by the natural gradient, that is by the $C^{-1}g$. [1] Intuitively, the natural gradient is more conservative and does not allow large variances. If the gradients highly disagree in one direction, one should not take the step. Thus, when-

[1]For more information on the natural gradient and how it relates to Newton's method and to the Hessian matrix we refer to (Le Roux et al., 2007).

ever we have computed a new gradient g_t we integrate its information and update the covariance at time step t by the following expression:

$$C_t = \gamma C_{t-1} + g_t g_t^T \tag{2}$$

where $C_0 = 0$, and γ is a parameter that controls how much older gradients are discounted. Now, each parameter θ_k has its own step size η_k and is updated by

$$\theta_{t+1} = \theta_t - \boldsymbol{\eta}_t \cdot g_t \tag{3}$$

The gain vector $\boldsymbol{\eta}_t$ serves as a diagonal conditioner and is simultaneously adapted with the meta-gain μ:

$$\eta_{t+1} \approx \eta_t \cdot \max(\frac{1}{2}, 1 - \mu g_{t+1} \cdot v_{t+1}) \tag{4}$$

where $v \in \Theta$ characterizes the long-term dependence of the system parameters on gain history over a time scale governed by the decay factor $0 \leq \lambda \leq 1$ and is iteratively updated by

$$v_{t+1} = \lambda v_t - \eta \cdot (g_t + \lambda C^{-1} v_t)\ . \tag{5}$$

To ensure a low computational complexity and a good stability of the computations, one can maintain a low rank approximation of C, (Le Roux et al., 2007). Using a per-parameter step-sizes considerably accelerates the convergence of stochastic natural gradient descent.

Putting everything together, we arrive at the lifted online learning for relational models That is, we randomly select a mini-batch, compute its gradient using lifted inference, and update the parameter vector. We iterate these steps until convergence. A common approach to test this considers the change of the parameter vector in the last l steps. If the change is small enough, we consider it as evidence of convergence. To simplify things, we may also simply fix the number of times we cycle through all mini-batches. This also allows to compare different methods.

3. Experimental Evaluation

We implemented our two algorithmic contributions of the lifted online learning for relational models (SMD and Tree-SMD) in Python. As a batch learning reference, we used *scaled conjugate gradient (SCG)* (Moller, 1993). SCG chooses the search direction and the step size by using information from the second order approximation. Inference that is needed as a subroutine for the learning methods was carried out by lifted belief propagation (LBP) (Kersting et al., 2009; Ahmadi et al., 2011). For evaluation, we computed the *conditional marginal log-likelihood* (CMLL) (Lee et al., 2007), which is defined with respect to marginal probabilities. All Experiments were conducted on a single cluster machine with 2.4 GHz and 64 GB of RAM.

Friends-and-Smokers MLN: In our first experiment we learned the parameters for the "Friends-and-Smokers" MLN (Singla & Domingos, 2008). We enriched the network by adding two clauses: if someone is stressed he is more likely to smoke and people having cancer should get medical treatment. For a given set of parameters we sampled 5 datasets from the joint distribution of the MLN with 10 persons. For each dataset we learned the parameters on this dataset and evaluated on the on the other four. The ground network of this MLN contains 380 factors and 140 variables. Fig. 2(left) shows the CMLL averaged over all of the 5 folds. We ran the lifted piecewiese learning with a batchsize of 10 and a step size of 0.2. Other parameters for SMD were chosen to be $\lambda = .99$, $\mu = 0.1$, and γ the discount for older gradients as 0.9.

As one can see, the lifted SMD has a steep learning curve and has already learned the parameters before seeing the mega example even once (indicated by the black vertical line), whereas SCG requires four passes over the entire training data to have a similar result in terms of CMLL. Moreover, as Fig 2(right) shows, piecewise learning greatly increases the lifting compared to batch learning, which essentially does not feature lifting at all.

CORA Entity Resolution MLN: In our second experiment we learned the parameters for the Cora entity resolution MLN, one of the standard datasets for relational learning. In the current paper, however, it is used in a non-standard, more challenging setting. For a set of bibliographies the Cora MLN has facts, e.g., about word appearances in the titles and in author names, the venue a paper appeared in, its title, etc. The task is to infer whether two entries in the bibliography denote the same paper (predicate *samePaper*) and two venues, titles and authors (*sameVenue*, *sameTitle*, and *sameAuthor* respectively) are the same. We sampled 20 bibliographies and extracted all facts corresponding to these bibliography entries. We constructed five folds then trained on four folds and tested on the fifth. We employed a transductive learning setting for this task. The MLN was parsed with all facts for the bibliographies from the five folds, i.e., the queries consisting of all four predicates were hidden for the test fold. The resulting ground network consisted of $36,390$ factors and $11,181$ variables. We learned the parameters using SCG, lifted SMD as well as Tree-SMD using relational treefinding with a threshold t of 0.9. The trees consisted of ten factors on average. So we updated with a batchsize of 100 for the trees and 1000 for standard pieces with a stepsize of 0.05, $\lambda = .99$, $\mu = 0.9$, $\gamma = 0.9$.

Fig. 2 (center) shows the averaged learning results. Again, online training does not need to see the whole mega-example; it has learned long before finishing one pass over the entire data. Moreover, building tree pieces considerably speeds-up the learning process. The Cora dataset contains a lot of strong de-

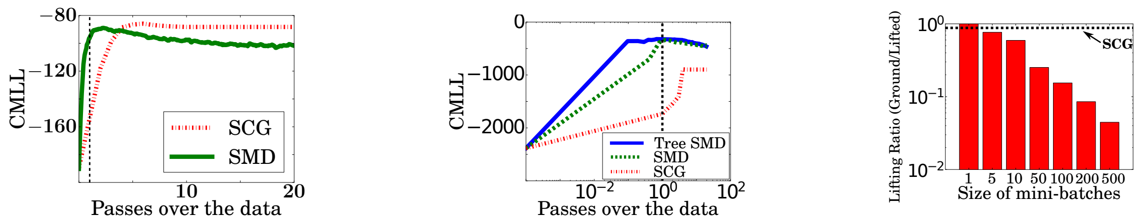

Figure 2. "Passes over mega-example" vs. Test-CMLL for the Friends-and-Smokers (left) and CORA (center) MLNs (the higher the better). Right: Lifting ratio for varying mini-batch size on Friends-and-Smokers MLN. (Best viewed in color)

pendencies which are all broken if we form one piece per factor. The trees on the other hand preserve parts of the local structure which significantly helps during learning. They convey a lot of additional information such that one obtains a better solution with less data.

4. Conclusions

In this paper, we have introduced the first lifted online training method for relational models. We employed the intuitively appealing idea of separately training pieces of the full model and combining the results in iteration and turned it into an online stochastic gradient method that processes one lifted piece after the other. This approach converges to the same quality solution over an order of magnitude faster, simply because unlike batch training it starts optimizing long before having seen the entire mega-example even once.

The stochastic relational gradient framework developed in the present paper puts many interesting research goals into reach. For instance, one should tackle one-pass relational learning by investigating different ways of gain adaption and scheduling of pieces for updates. One should also investigate budget constraints on both the number of examples and the computation time per iteration. At massive scales parallel and distributed algorithms for training are essential.

Acknowledgements: The authors thank the reviewers for the helpful comments. BA and KK were supported by the Fraunhofer ATTRACT fellowship STREAM and the European Commission under contract number FP7-248258-First-MM. SN gratefully acknowledges the support of the DARPA Machine Reading Program under AFRL prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of DARPA, AFRL, or the US government.

References

Ahmadi, B., Kersting, K., and Sanner, S. Multi-Evidence Lifted Message Passing, with Application to PageRank and the Kalman Filter. In *IJCAI*, 2011.

Ahmadi, B., Kersting, K., and Natarajan, S. Lifted online training of relational models with stochastic gradient methods. In *ECML-PKDD*, 2012.

Amari, S. Natural gradient works efficiently in learning. *Neural Comput.*, 10:251–276, February 1998.

Huynh, T. and Mooney, R. Online max-margin weight learning for markov logic networks. In *SDM*, 2011.

Kersting, K., Ahmadi, B., and Natarajan, S. Counting belief propagation. In *UAI*, Montreal, Canada, 2009.

Khot, T., Natarajan, S., Kersting, K., and Shavlik, J. Learning markov logic networks via functional gradient boosting. In *ICDM*, 2011.

Kok, S. and Domingos, P. Learning Markov logic network structure via hypergraph lifting. In *ICML*, 2009.

Kok, S. and Domingos, P. Learning Markov logic networks using structural motifs. In *ICML*, 2010.

Le Roux, N., Manzagol, P.-A., and Bengio, Yoshua. Topmoumoute online natural gradient algorithm. In *NIPS*, 2007.

Lee, S.-I., Ganapathi, V., and Koller, D. Efficient structure learning of Markov networks using L1-regularization. In *NIPS*, 2007.

Moller, M. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 1993.

Richards, B.L. and Mooney, R.J. Learning relations by pathfinding. In *AAAI*, pp. 50–55, 1992.

Richardson, M. and Domingos, P. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

Schraudolph, N. and Graepel, T. Combining conjugate direction methods with stochastic approximation of gradients. In *AISTATS*, pp. 7–13, 2003.

Singla, P. and Domingos, P. Lifted First-Order Belief Propagation. In *AAAI*, 2008.

Sutton, C. and McCallum, A. Piecewise training for structured prediction. *Machine Learning*, 2009.

Vishwanathan, S. V. N., Schraudolph, Nicol N., Schmidt, Mark W., and Murphy, Kevin P. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML*, 2006.