Lifted Online Training of Relational Models with Stochastic Gradient Methods

Babak Ahmadi[1], Kristian Kersting[1,2,3], and Sriraam Natarajan[3]

[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, which processes each lifted piece one after the other. On several datasets, the resulting optimizer 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.

1 Introduction

Statistical relational models, see [1, 2] for overviews, have recently gained popularity in the machine learning and AI communities since they 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.

One attractive avenue to scale relational learning is based on lifted message-passing approaches [3, 4]. They have rendered large, previously intractable probabilistic inference problems quickly (often approximately) solvable by employing symmetries to handle whole sets of indistinguishable random variables. Still, in most situations training relational models will not benefit from lifting:

(Limitation 1) *Symmetries within a model easily break since variables become correlated by virtue of depending asymmetrically on evidence.*

Because of this, lifting produces new models that are often not far from propositionalized, therefore canceling the benefits of lifting for training. Moreover, in relational learning we often face a single mega-example [5] only, a single large set of inter-connected facts. Consequently, many if not all standard statistical learning methods do not naturally carry over to the relational case. Consider e.g. stochastic gradient methods. Similar to the perceptron method [6], stochastic gradient descent algorithms update the weight vector in an online setting. We essentially assume that the training examples are given one at a time. The algorithms examine the current training example 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. Empirically, they are even often found to be more resilient to errors made when approximating the gradient. Unfortunately, stochastic gradient methods do not naturally carry over to the relational cases:

> **(Limitation 2)** *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. This overcomes **limitation 1** by breaking long-range dependencies and allows one — as we will show — to exploit lifting across the local training tasks. It also paves the way for online training — as we will show — of relational models since we can treat (mini-batches of) pieces as training examples and process one piece after the other, hence overcoming **limitation 2**. Based on this insight, we develop our main algorithmic contribution: the first lifted online training approach for relational models using a 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. After touching upon related work, 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 Related Work

Our work aims at combining stochastic gradient methods for online training, relational learning, and lifted inference hence is related to several lines of research.

Local training is well known for propositional graphical models. Besag [7] presented a pseudolikelihood (PL) approach for training an Ising model with a rectangular array of variables. PL, however, tends to introduce a bias and is not necessarily a good approximation of the true likelihood with a smaller number of samples. In the limit, however, the maximum pseudolikelihood coincides with that of the true likelihood [8]. Hence, it is a very popular method for training models such as Conditional Random Fields (CRF) where the normalization can become intractable while PL requires normalizing over only one node. An alternative approach is to decompose the factor graph into tractable subgraphs (or pieces) that are trained independently [9], as also follows in the present paper. This *piecewise training* can be understood as approximating the exact likelihood using a propagation algorithm such as BP. Sutton and McCallum [9] also combined the two ideas of PL and piecewise training to propose piecewise pseudolikelihood (PWPL) which inspite of being a double approximation has the benefit of being accurate like piecewise and scales well due to the use of PL. Another intuitive approach is to compute approximate marginal distributions using a global propagation algorithm like BP, and simply substitute the resulting beliefs into the exact ML gradient [10], which will result in approximate partial derivatives. Similarly, the beliefs can also be used by a sampling method such as MCMC where the true marginals are approximated by running an MCMC algorithm for a few iterations. Such an approach is called constructive divergence [11] and is a popular method for training CRFs.

All the above methods were originally developed for propositional data while real-world data is inherently noisy and relational. Statistical Relational Learning (SRL) [1, 2] deals with uncertainty and relations among objects. The advantage of relational models is that they can succinctly represent probabilistic dependencies among the attributes of different related objects leading to a compact representation of learned models. While relational models are very expressive, learning them is a computationally intensive task. Recently, there have been some advances in learning SRL models, especially in the case of Markov Logic Networks [12–14]. Algorithms based on functional-gradient boosting [15] have been developed for learning SRL models such as Relational Dependency Networks [16], and Markov Logic Networks [14]. Piecewise learning has also been pursued already in SRL. For instance, the work by Richardson and Domingos [17] used pseudolikelihood to approximate the joint distribution of MLNs which is inspired from the local training methods mentioned above. 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. They cannot easily deal with missing information. To do so, algorithms based on classical EM [18] have been developed for ProbLog, CP-logic, PRISM, probabilistic relational models, Bayesian logic programs [19–23], among others, as well as gradient-based approaches for relational models with complex combining rules [24, 25]. All these approaches, however, assume a *batch* learning setting; they do not update the parameters until the entire data has been scanned. In the presence of large amounts of data such as relational data, the above method

can be wasteful. Stochastic gradient methods as considered in the present paper, on the other hand, are online and scale sub-linearly with the amount of training data, making them very attractive for large data sets. Only Huynh and Mooney [26] 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. They, however, do not employ lifted inference for training and also make the closed-world assumption.

3 Markov Logic Networks

We develop our lifted online training method within the framework of Markov logic networks [17] but would like to note that it naturally carries over to other relational frameworks. A Markov logic network (MLN) is defined by a set of first-order formulas (or clauses) 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, i.e. the free variables in the predicates of the formulas F_i are bound to be constants in C, to define a Markov network. This ground Markov network contains a binary node for each possible grounding of each predicate, and a feature for each grounding f_k of each formula. 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 Logic networks can be formulated as follows. Given a set of training instances $D = \{D_1, D_2, \ldots D_M\}$ each constisting of an assignment to the variables in X the goal is to output a parameter vector θ specifying a weight for each $F_i \in F$. Typically, however, a single mega-example [5] is only given, a single large set of inter-connected facts. For the sake of simplicity we will sometimes denote the mega-example simply as E. To train the model, we can seek to maximize the log-likelihood function $\log P(D \mid \theta)$ given by $\ell(\theta, D) = \frac{1}{n} \sum_D \log P_\theta(X = x_{D_n})$. The likelihood, however, is computationally hard to obtain. A widely-used alternative is to maximize the pesudo-log-likelihood instead i.e., $\log P^*(X = x \mid \theta) = \sum_{l=1}^{n} \log P_\theta(X = x_l | MB_x(X_l))$ where $MB_x(X_l)$ is the state of the Markov blanket of X_l in the data, i.e. the assignment of all variables neighboring X_l. In this paper, we resort to likelihood maximization. No matter which objective function is used, one typically runs a gradient-descent to train the model. That is, we start with some initial parameters θ_0 — typically initialized to be zero or at random around zero — and update the parameter vector using $\theta_{t+1} = \theta_t - \eta_t \cdot g_t$. Here g_t denotes the gradient of the likelihood function and is given by:

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

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. This inference step must be performed at every step of the gradient process. In the case of partially observed data we cannot simply read-off the feature counts in the empirical data and have to perform inference there as well. Consequently, there is a close interaction between the training approach and the inference method employed for training.

4 Lifted Online Training

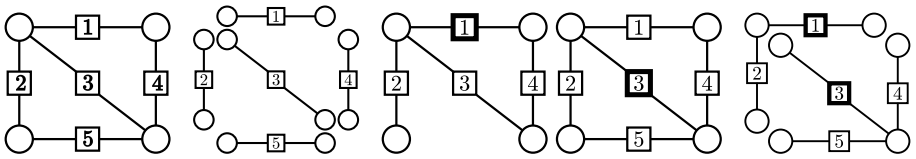
Lifted Belief propagation (LBP) approaches [27, 4] have recently drawn a lot of attention as they render large previously intractable models quickly solvable by exploiting symmetries. Such symmetries are commonly found in first-order and relational probabilistic models that combine aspects of first-order logic and probability. Instantiating all ground atoms from the formulae in such models induces a standard graphical model with symmetric, repeated potential structures for all grounding combinations. To exploit the symmetries, LBP approaches 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. When learning parameters of a given model for a given set of observations, however, the presence of evidence on the variables mostly destroys the symmetries. This makes lifted approaches virtually of no use if the evidence is non symmetrical.

In the fully observed case, this may not be a major obstacle since 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 rarely gets all of the possible tests. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and typically involves nonlinear, iterative optimization and multiple calls to a relational inference engine as subroutine.

Since efficient lifted inference is troublesome in the presence of partial evidence and most lifted approaches basically fall back to the ground variants 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.

4.1 Piecewise Shattering

In piecewise training, we decompose the mega-example and its corresponding factor graph into tractable but not necessarily disjoint subgraphs (or pieces) $\mathcal{P} = \{p_1, \ldots, p_k\}$ that are trained independently [28]. Intuitively, the pieces turn the single mega-example into a set of many training examples and hence pave the



(a) Orig. model (b) Depth $d = 0$ (c) Depth $d = 1$ for f_1 and f_3 (d) Trees $d = 1$

Fig. 1. Schematic factor-graph depiction of the difference between likelihood (a), standard piecewise (b,c) and treewise training (d). Likelihood training considers the whole mega-example, i.e., it performes inference on the complete factor graph induced over the mega-example. Here, circles denote random variables, and boxes denote factors. Piecewise training normalizes over one factor at a time (b) or higher-order, complete neighbourhoods of a factor (c) taking longer dependcies into account, here shown factors f_1 and f_3. Treewise training (d) explores the spectrum between (b) and (c) in that it also takes longer dependecies into account but does not consider complete higher neighbourhoods; shown for tree features for factors f_1 and f_3. In doing so it balances complexity and accuracy of inference.

way for online training. This is a reasonable idea since in many applications, the local information in each factor alone 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 separate 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 separate piece for each factor. Intuitively, however, this discards dependencies of the model parameters when we decompose the mega-example into pieces. 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.

4.2 Relational Tree Shattering

Assume that the mega-example has been turned into a single factor graph for performing inference, cf. Fig. 1**(a)**. A factor graph is a bipartite graph and contains nodes representing random variables (denoted by circles) and factors (squares). It explicitly represents the factorization of the graphical model and there is an edge between a factor f_k and a node i iff variable X_i appears in f_k. Now, 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

Algorithm 1: RelTreeFinding: Relational Treefinding

Input: Set of clauses **F**, a mega example **E**, depth **d**, and discount $t \in [0, 1]$
Output: Set of tree pieces **T**

```
// Tree-Pattern Finding
```
Initialize the dictionary of tree patterns to be empty, i.e., $P = \emptyset$;
for each *clause* $F_i \in F$ **do**
  Select a random ground instance f_j of F_i in E;
  Initialize tree pattern for F_i, i.e., $P_i = \{f_j\}$;
  `// perform random walk in a breath-first manner starting in` f_j
  for f_k = `BFS.`*next()* **do**
    if *current_depth* $> d$ **then** break;
    sample p uniformly from $[0, 1]$;
    if $p > t^{|P_i|}$ **or** f_k *would induce a cycle* **then**
      skip branch rooted in f_k in BFS ;
    else
      add f_k to P_i;
  Variablilize P_i and add it to dictonary P;
```
// Construct tree-based pieces using the relational tree patterns
```
for each $f_j \in$ **E do**
  Find $P_k \in P$ matching f_j, i.e., the tree pattern rooted in the clause F_k corresponding to factor f_j;
  Unify P_k with f_j to obtain piece T_j and add T_j to T ;
return T;

piecewise model as shown in Fig. 1**(b)**, i.e. each factor is isolated in a separate 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 network 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 since 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. Formally, a tree is defined as a set of factors such that for any two factors f_1 and f_n in the set, there exists one and only one ordering of (a subset of) factors in the set f_1, f_2 , $\ldots$ f_n such that f_i and f_{i+1} share at least one variable, i.e. there are no loops. 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 [29].

The relational treefinding is summarized in Alg. 1. For a given set of Clauses F and a mega example E the algorithm starts off by constructing a tree pattern

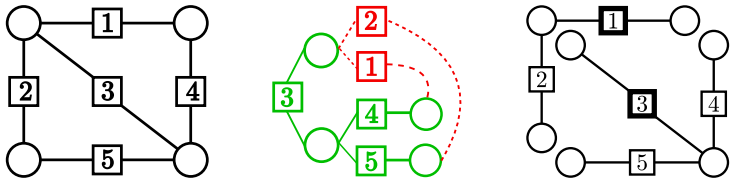


Fig. 2. Illustration of tree shattering: from the original model (left) we compute a tree piece (right). Starting from factor f_3, we randomly follow the tree-structured "unrolling"' of the graphical model rooted at f_3. Green shows that the factor has been included in the random walk while all red factors have been discarded. This results in the tree pattern for f_3 shown on the right hand side. A similar random walk generated the other shown tree pattern for f_1.

for each clause F_i **(lines 1-12)**. Therefore, it first selects a random ground instance f_j **(line 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 **(line 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, or the factor would induce a cycle, 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_j we apply the pattern P_k of clause F_k which generated the factor **(lines 13-15)**.

These *tree-based pieces* can balance efficiency and quality of the parameter estimation well. Reconsider the example from Fig. 1. Fig. 2 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 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 $p \in [0, 1]$. It was small enough, e.g. $p = 0.3 < 0.9^1$ so f_4 is added to the tree. For f_2 we sample $p = 0.85 > 0.9^2$ so f_2 and its branch are discarded. For f_1 we sample $p = 0.5 < 0.9^2$ so f_1 could be added. If we added f_1, however, it would together with f_3 and f_4 form a cycle, so its branch is discarded. For f_5 we sample $p = 0.4 < 0.9^2$ so it is added to the tree. Note that now 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 $A(\Theta)$. To see, this, we first write the original parameter vector Θ as a mixture of parameter vectors $\Theta(T_t)$ induced by the tractable subgraphs. For each edge in our mega-example E, we add a non-spanning tree T_t which contains all the original vertices but only the edges present in t. With each tree T_t we associate an exponential parameter vector $\Theta(T_t)$. Let μ be a strictly positive probability distribution over the tractable subgraphs, such that the original parameter vector Θ can be written as a combination of per-tree-clause parameter vectors

$$\Theta = \sum_F \sum_t \mu_{t,F} \Theta(T_t) \ ,$$

where we have expressed parameter sharing among the ground instance of the clauses. Now using Jensen's inequality, we can state the following upper bound to the log partition function:

$$A(\Theta) = A(\sum_F \sum_t \mu_{t,F} \Theta(T_t)) = A(\sum_t \mu_t \Theta(T_t)) \leq \sum_t \mu_t A(\Theta(T_t)) \qquad (2)$$

with $\mu_t = \sum_F \mu_{t,F}$. Since the $\mu_{t,F}$ are convex, the μ_t are convex, too, and applying Jensen's inequality is safe. So we can follow Sutton and McCallum's [9] arguments. Namely, for tractable subgraphs and a tractable number of models the right-hand side of (2) can be computed efficiently. Otherwise it forms an optimization problem, which according to [30] can be interpreted as free energy and depends on a set of marginals and edge appearance probabilities, in our case the probability that an edge appears in a tree, i.e. is visited in the random walk. Also, it is easy to show that pieces of depth 0 are an upper bound to this bound since, we can apply Jensen's inequality again when breaking the trees into independent pathes from the root to the leaves.

Now, we show how to turn this upper bound into a lifted online training for relational models.

4.3 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

$$\sum\nolimits_i \frac{1}{\#_i} \frac{\partial \ell(\theta, D_i)}{\partial \theta_k}, \qquad (3)$$

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 example, let g_i be a gradient over the the mini-batch D_i. 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 together. For example, if for a model with 4 clauses the single piece mini-batch D_i has counts $(1, 3, 0, 2)$ the gradient is normalized by the respective counts. If the mini-batch, however, has an additional piece with counts $(0, 2, 1, 0)$ we normalize by the sum, i.e. $(1, 5, 1, 2)$.

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 [31] 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 [32]. However, while SMD is very efficient in Euclidian spaces, Amari [33] 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 $C^{-1}g$. 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, whenever we have computed a new gradient g_t we integrate its information and update the covariance at time step t by the following expression:

$$C_t = \gamma C_{t-1} + g_t g_t^T \tag{4}$$

where $C_0 = 0$, and γ is a parameter that controls how much older gradients are discounted. Now, let each parameter θ_k have its own step size η_k. We update the parameter b

$$\theta_{t+1} = \theta_t - \boldsymbol{\eta}_t \cdot g_t \tag{5}$$

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

$$\eta_{t+1} = \eta_t \cdot \exp(-\mu g_{t+1} \cdot v_{t+1}) \approx \eta_t \cdot \max(\frac{1}{2}, 1 - \mu g_{t+1} \cdot v_{t+1}) \tag{6}$$

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

$$v_{t+1} = \lambda v_t - \eta \cdot (g_t + \lambda C^{-1} v_t) \; . \tag{7}$$

Algorithm 2: Lifted Online Training of Relational Models

Input: Markov Logic Network $\mathbf{M}$, mega-example E, decay factors t, γ, and λ
Output: Parameter vector θ

```
// Generate mini-batches
```
1 Generate set of tree pieces $\mathcal{P}$ using RelTreeFinding;
2 Randomly form mini-batches $\mathcal{B} = \{B_1, \ldots, B_m\}$ each consisting of l pieces;

```
// Peform lifted stochastic meta-descent
```
3 Initialize θ and v_0 with zeros and the covariance matrix $\mathbf{C}$ to the zero matrix;
4 **while** *not converged* **do**
5 Shuffle mini-batches $\mathcal{B}$ randomly;
6 **for** $i = 1, 2, \ldots, m$ **do**
7 Compute gradient g for B_i using lifted belief propagation;
8 Update covariance matrix $\mathbf{C}$ using (4) or some low-rank variant;
9 Update parameter vector θ using (5) and the involved equations;
10 **return** θ;

To ensure a low computational complexity and a good stability of the computations, one can maintain a low rank approximation of C, see [34] for more details. Using 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 as summarized in Alg. 2. That is, we form mini-batches of tree pieces **(lines 1-2)**. After initialization **(lines 3-4)**, we then perform lifted stochastic meta-descent **(lines 5-9)**. That is, we randomly select a mini-batch, compute its gradient using lifted inference, and update the parameter vector. Note that pieces and mini-batches can also be computed on the fly and thus its construction be interweaved with the parameter update. We iterate these steps until convergence, e.g. by considering 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.

5 Experimental Evaluation

Our intention here is to investigate the following questions: **(Q1)** Can we efficiently train relational models using stochastic gradients? **(Q2)** Are there symmetries within mini-batches that result in lifting? **(Q3)** Can relational treefinding produce pieces that balance accuracy and lifting well? **(Q4)** Is it even possible to achieve one-pass relational training?

To this aim, we implemented lifted online learning for relational models in Python. As a batch learning reference, we used *scaled conjugate gradient (SCG)* [35]. 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) [4, 36]. For evaluation, we computed the *conditional marginal*

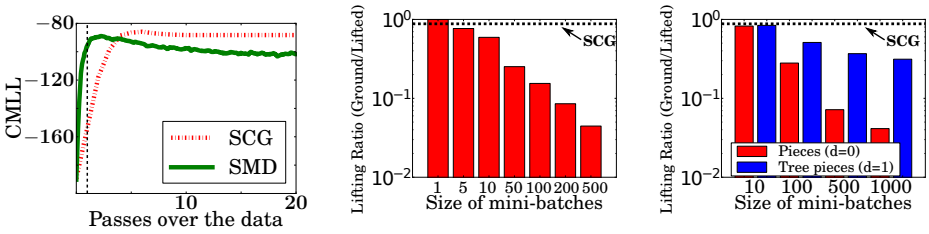


Fig. 3. "Passes over mega-example" vs. Test-CMLL for the Friends-and-Smokers **(left)** (the higher the better). lifted online learning has already learned before seeing the mega example even once (black vertical line). **(center)** Benefit of local training for lifting. Lifting ratio for varying mini-batch size versus the full batch model on the Friends-and-Smokers MLN. Clearly for a batch size of 1 there is no lifting but with larger mini-batch sizes there is more potential to lift the pieces within each batch; the size can be an order of magnitude smaller. **(right)** Lifting ratio for standard pieces vs. tree pieces on the Voting MLN. Due to rejoining of pieces, additional symmetries are broken and the lifting potential is smaller. However, the sizes of the models per mini-batch still gradually decrease with larger mini-batch sizes. (Best viewed in color)

log-likelihood (CMLL) [10], which is defined with respect to marginal probabilities. More precisely, we first divide the variables into two groups: X_{hidden} and $X_{observed}$. Then, we compute $\text{CMLL}(E) = \sum_{X \in X_{hidden}} \log P(X|X_{observed})$ for the given mega-example. To stabilize the metric, we divided the variables into four groups and calculated the average CMLL when observing only one group and hiding the rest. All experiments were conducted on a single machine with 2.4 GHz and 64 GB of RAM.

(Q1, Q2) Friends-and-Smokers MLN: In our first experiment we learned the parameters for the "Friends-and-Smokers" MLN [27], which basically defines rules about the smoking behaviour of people, how the friendship of two people influences whether a person smokes or not, and that a person is more likely to get cancer if he smokes. 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 dataset from the joint distribution of the MLN with 10 persons. For each dataset we learned the parameters on this dataset and evaluated on the other four. The ground network of this MLN contains 380 factors and 140 variables. The batchsize was 10 and we used a stepsize of 0.2. Fig. 3(left) shows the CMLL averaged over all of the 5 folds. We ran the lifted piecewise 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. Note that we learned the models without stopping criterion and for a fixed number of passes over the data thus the CMLL on the

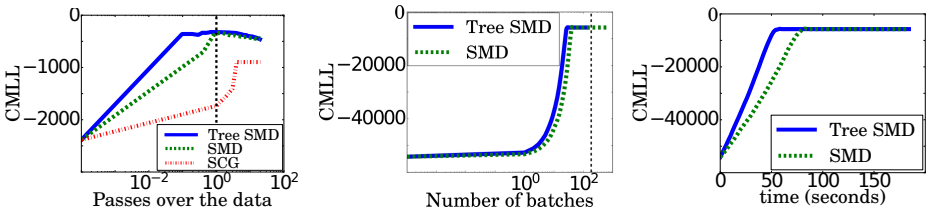


Fig. 4. Experimental results. From left to right, "passes over mega-example" vs. Test-CMLL for the CORA and "number of batches" vs. Test-CMLL for the Wumpus MLNs (the higher the better). The last graph on the right-hand-side shows the runtime vs. CMLL on the Wumpus MLN. As one can see, lifted online learning has already converged before seeing the mega example even once (black vertical line). For the Wumpus MLN, SCG did not converge within 72 hours. (Best viewed in color)

test data can decrease. SCG on the other hand requires four passes over the entire training data to have a similar result in terms of CMLL. Thus **Q1** can be answered affirmatively. Moreover, as Fig 3(center) shows, piecewise learning greatly increases the lifting compared to batch learning, which essentially does not feature lifting at all. Thus, **Q2** can be answered affirmatively.

(Q2,Q3) Voting MLN: To investigate whether tree pieces although more complex can still yield lifting, we considered the Voting MLN from the Alchemy repository. The network contains 3230 factors and 3230 variables. Note that it is a propositional Naive Bayes (NB) model. Hence, depth 0 pieces will yield greater lifting but hamper information flow among attributes if the class variable is unobserved. Tree pieces intuitively couple depth 0 hence will indeed yield lower lifting ratios. However, with larger mini-batches they should still yield higher lifting than the batch case. This is confirmed by the experimental results summarized in Fig 3(right). Thus, **Q3** can be answered affirmatively.

(Q3,Q4) 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 now to infer whether two entries in the bibliography denote the same paper (predicate *samePaper*), two venues are (*sameVenue*), two titles are the same (*sameTitle*), and whether two authors are the same (*sameAuthor*). 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 were hidden for the test fold. The query consisted of all four predicates (*sameAuthor*,*samePaper*,*sameBib*, *sameVenue*). The resulting ground network consisted of $36,390$ factors and $11,181$ variables. We learnt the parameters using SCG, lifted stochastic meta-descent with standard pieces as well as pieces using relational treefinding with a threshold t of

0.9. The trees consisted of around 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. Furthermore, other parameters were chosen to be $\lambda = .99$, $\mu = 0.9$, and $\gamma = 0.9$. Fig. 4(left) shows the averaged learning results for this entity resolution task. Again, online training does not need to see the whole mega-example; it has learned long before finishing one pass over the entire data. Thus, **(Q4)** can be answered affirmatively.

Moreover, Fig. 4 also shows that by building tree pieces one can considerably speed-up the learning process. They convey a lot of additional information such that one obtains a better solution with a smaller amount of data. This is due to the fact that the Cora dataset contains a lot of strong dependencies 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. Thus, **(Q3)** can be answered affirmatively.

(Q3,Q4) Lifted Imitation Learning in the Wumpus Domain: To further investigate **(Q3)** and **(Q4)**, we considered imitation learning in a relational domain for a Partially Observed Markov Decision Process (POMDP). We created a simple version of the Wumpus task where the location of Wumpus is partially observed. We used a 5×5 grid with a Wumpus placed in a random location in every training trajectory. The Wumpus is always surrounded by stench on all four sides. We do not have any pits or breezes in our task. The agent can perform 8 possible actions: 4 move actions in each direction and 4 shoot actions in each direction. The agent's task is to move to a cell so that he can fire an arrow to kill the Wumpus. The Wumpus is not observed in all the trajectories although the stench is always observed. Trajectories were created by real human users who play the game. The resulting network contains 182400 factors and 4469 variables. We updated with a batchsize of 200 for the trees and 2000 for standard pieces with a stepsize of 0.05. As for the cora dataset used $\lambda = .99$, $\mu = 0.9$, and $\gamma = 0.9$.

Figure 4 shows the result on this dataset for lifted SMD with standard pieces as well as pieces using relational treefinding with a threshold t of 0.9. For this task, SCG did not converge within 72 hours. Note that this particular network has a complex structure with lots of edges and large clauses. This makes inference on the global model intractable. Fig. 4 (center) shows the learning curve for the total number of batches seen as well as the total time needed for one pass over the data (right). As one can see, tree pieces actually yield faster convergence, again long before having seen the dataset even once. Thus, **(Q3)** and **(Q4)** can be answered affirmatively.

Taking all experimental results together, all questions **Q1**-**Q4** can be clearly answered affirmatively.

6 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. We showed that this approach can be justified as maximizing a loose bound on the log likelihood and that it 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. In general, relational problems can easily involve models with millions of random variables. At such massive scales, parallel and distributed algorithms for training are essential to achieving reasonable performance.

Acknowledgements: The authors thank the reviewers for their helpful comments. BA and KK were supported by the Fraunhofer ATTRACT fellowship STREAM and by 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

1. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. The MIT Press (2007)
2. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming. Lecture Notes in Computer Science. Springer (2008)
3. Singla, P., Domingos, P.: Lifted First-Order Belief Propagation. In: AAAI. (2008)
4. Kersting, K., Ahmadi, B., Natarajan, S.: Counting belief propagation. In: UAI, Montreal, Canada (2009)
5. Mihalkova, L., Huynh, T., Mooney, R.: Mapping and revising markov logic networks for transfer learning. In: AAAI. (2007) 608–614
6. Rosenblatt, F.: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan (1962)
7. Besag, J.: Statistical Analysis of Non-Lattice Data. Journal of the Royal Statistical Society. Series D (The Statistician) **24**(3) (1975) 179–195
8. Winkler, G.: Image Analysis, Random Fields and Dynamic Monte Carlo Methods. Springer-Verlag (1995)
9. C.Sutton, Mccallum, A.: Piecewise training for structured prediction. Machine Learning **77**(2–3) (2009) 165–194
10. Lee, S.I., Ganapathi, V., Koller, D.: Efficient structure learning of Markov networks using L1-regularization. In: NIPS. (2007)
11. Hinton, G.: Training products of experts by minimizing contrastive divergence. Neural Computation **14** (2002)
12. Kok, S., Domingos, P.: Learning Markov logic network structure via hypergraph lifting. In: ICML. (2009)

13. Kok, S., Domingos, P.: Learning Markov logic networks using structural motifs. In: ICML. (2010)
14. Khot, T., Natarajan, S., Kersting, K., Shavlik, J.: Learning markov logic networks via functional gradient boosting. In: ICDM. (2011)
15. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics (2001) 1189–1232
16. Natarajan, S., Khot, T., Kersting, K., Guttmann, B., Shavlik, J.: Gradient-based boosting for statistical relational learning: The relational dependency network case. Machine Learning (2012)
17. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1-2) (2006) 107–136
18. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society **B.39** (1977)
19. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. J. Artif. Intell. Res. (JAIR) **15** (2001) 391–454
20. Kersting, K., De Raedt, L.: Adaptive bayesian logic programs. In: ILP. (2001)
21. Getoor, L., Friedman, N., Koller, D., Taskar, B.: Learning probabilistic models of link structure. Journal of Machine Learning Research **3** (2002) 679–707
22. Thon, I., Landwehr, N., De Raedt, L.: Stochastic relational processes: Efficient inference and applications. Machine Learning **82**(2) (2011) 239–272
23. Gutmann, B., Thon, I., De Raedt, L.: Learning the parameters of probabilistic logic programs from interpretations. In: ECML-PKDD. (2011) 581–596
24. Natarajan, S., Tadepalli, P., Dietterich, T.G., Fern, A.: Learning first-order probabilistic models with combining rules. Annals of Mathematics and AI (2009)
25. Jaeger, M.: Parameter learning for Relational Bayesian networks. In: ICML. (2007)
26. Huynh, T., Mooney, R.: Online max-margin weight learning for markov logic networks. In: SDM. (2011)
27. Singla, P., Domingos, P.: Lifted first-order belief propagation. In: AAAI. (2008)
28. Sutton, C., McCallum, A.: Piecewise training for structured prediction. Machine Learning **77**(2–3) (2009) 165–194
29. Richards, B., Mooney, R.: Learning relations by pathfinding. In: AAAI. (1992)
30. Wainwright, M., Jaakkola, T., Willsky, A.: A new class of upper bounds on the log partition function. In: UAI. (2002) 536–543
31. Schraudolph, N., Graepel, T.: Combining conjugate direction methods with stochastic approximation of gradients. In: AISTATS. (2003) 7–13
32. Vishwanathan, S.V.N., Schraudolph, N.N., Schmidt, M.W., Murphy, K.P.: Accelerated training of conditional random fields with stochastic gradient methods. In: In ICML. (2006) 969–976
33. Amari, S.: Natural gradient works efficiently in learning. Neural Comput. **10** (1998) 251–276
34. Le Roux, N., Manzagol, P.A., Bengio, Y.: Topmoumoute online natural gradient algorithm. In: NIPS. (2007)
35. Mller, M.: A scaled conjugate gradient algorithm for fast supervised learning. NEURAL NETWORKS **6**(4) (1993) 525–533
36. Ahmadi, B., Kersting, K., Sanner, S.: Multi-Evidence Lifted Message Passing, with Application to PageRank and the Kalman Filter. In: IJCAI. (2011)