Efficient Weight Learning for Markov Logic Networks

Daniel Lowd and Pedro Domingos

Department of Computer Science and Engineering
University of Washington, Seattle WA 98195-2350, USA
{lowd,pedrod}@cs.washington.edu

Abstract. Markov logic networks (MLNs) combine Markov networks and first-order logic, and are a powerful and increasingly popular representation for statistical relational learning. The state-of-the-art method for discriminative learning of MLN weights is the voted perceptron algorithm, which is essentially gradient descent with an MPE approximation to the expected sufficient statistics (true clause counts). Unfortunately, these can vary widely between clauses, causing the learning problem to be highly ill-conditioned, and making gradient descent very slow. In this paper, we explore several alternatives, from per-weight learning rates to second-order methods. In particular, we focus on two approaches that avoid computing the partition function: diagonal Newton and scaled conjugate gradient. In experiments on standard SRL datasets, we obtain order-of-magnitude speedups, or more accurate models given comparable learning times.

1 Introduction

Statistical relational learning (SRL) focuses on domains where data points are not i.i.d. (independent and identically distributed). It combines ideas from statistical learning and inductive logic programming, and interest in it has grown rapidly in recent years [6]. One of the most powerful representations for SRL is Markov logic, which generalizes both Markov random fields and first-order logic [16]. Representing a problem as a Markov logic network (MLN) involves simply writing down a list of first-order formulas and learning weights for those formulas from data. The first step is the task of the knowledge engineer; the second is the focus of this paper.

Currently, the best-performing algorithm for learning MLN weights is Singla and Domingos' voted perceptron [19], based on Collins' earlier one [3] for hidden Markov models. Voted perceptron uses gradient descent to approximately optimize the conditional likelihood of the query atoms given the evidence. Weight learning in Markov logic is a convex optimization problem, and thus gradient descent is guaranteed to find the global optimum. However, convergence to this optimum may be extremely slow. MLNs are exponential models, and their sufficient statistics are the numbers of times each clause is true in the data. Because this number can easily vary by orders of magnitude from one clause to another, a learning rate that is small enough to avoid divergence in some weights is too small for fast convergence in others. This is an instance of the well-known problem of ill-conditioning in numerical optimization, and many candidate solutions for it exist [13]. However, the most common ones are not easily applicable to MLNs because of the nature of the function being optimized. As in Markov random fields, computing the likelihood in MLNs requires computing the partition function, which is generally intractable. This makes it difficult to apply methods that require performing line searches, which involve computing the function as well as its gradient. These include most conjugate gradient and quasi-Newton methods (e.g., L-BFGS). Two exceptions to this are scaled conjugate gradient [12] and Newton's method with a diagonalized Hessian [1]. In this paper we show how they can be applied to MLN learning, and verify empirically that they greatly speed up convergence. We also obtain good results with a simpler method: per-weight learning rates, with a weight's learning rate being the global one divided by the corresponding clause's empirical number of true groundings.

Voted perceptron approximates the expected sufficient statistics in the gradient by computing them at the MPE state (i.e., the most likely state of the non-evidence atoms given the evidence ones, or most probable explanation). Since in an MLN the conditional distribution can contain many modes, this may not be a good approximation. Also, using second-order methods requires computing the Hessian (matrix of second-order partial derivatives), and for this the MPE approximation is no longer sufficient. We address both of these problems by instead computing expected counts using MC-SAT, a very fast Markov chain Monte Carlo (MCMC) algorithm for Markov logic [15].

The remainder of this paper is organized as follows. In Section 2 we briefly review Markov logic. In Section 3 we present several algorithms for MLN weight learning. We compare these algorithms empirically on real-world datasets in Section 4, and conclude in Section 5.

2 Markov Logic

A Markov logic network (MLN) consists of a set of first-order formulas and their weights, $\{(w_i, f_i)\}$. Intuitively, a formula represents a noisy relational rule, and its weight represents the relative strength or importance of that rule. Given a finite set of constants, we can instantiate an MLN as a Markov random field (MRF) in which each node is a grounding of a predicate (atom) and each feature is a grounding of one of the formulas (clauses). This leads to the following joint probability distribution for all atoms:

$$P(\mathrm{X} = \mathrm{x}) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$$

where n_i is the number of times the ith formula is satisfied by the state of the world x and Z is a normalization constant, required to make the probabilities of all worlds to sum to one.

The formulas in an MLN are typically specified by an expert, or they can be obtained (or refined) by inductive logic programming or MLN structure learning [10]. Many complex models, and in particular many non-i.i.d. ones, can be very compactly specified using MLNs.

Exact inference in MLNs is intractable. Instead, we can perform approximate inference using Markov chain Monte Carlo (MCMC), and in particular Gibbs sampling [7]. However, when weights are large convergence can be very slow, and when they are infinite (corresponding to deterministic dependencies) ergodicity breaks down. This remains true even for more sophisticated alternatives like simulated tempering. A much more efficient alternative, which also preserves ergodicity in the presence of determinism, is the MC-SAT algorithm, recently introduced by Poon and Domingos [15]. MC-SAT is a "slice sampling" MCMC algorithm that uses a modified satisfiability solver to sample from the slice. The solver is able to find isolated modes in the distribution very efficiently, and as a result the Markov chain mixes very rapidly. The slice sampling scheme ensures that detailed balance is (approximately) preserved. In this paper we use MC-SAT for inference.

3 Weight Learning for MLNs

Given a set of formulas and a database of atoms, we wish to find the formulas' maximum *a posteriori* (MAP) weights, i.e., the weights that maximize the product of their prior probability and the data likelihood. Since optimization is typically posed as function minimization, we will equivalently minimize the negative log-likelihood.

Richardson and Domingos [16] originally proposed learning weights generatively using pseudo-likelihood [2]. Pseudo-likelihood is the product of the conditional likelihood of each variable given the values of its neighbors in the data. While efficient for learning, it can give poor results when long chains of inference are required at query time. Singla and Domingos [19] showed that pseudo-likelihood is consistently outperformed by discriminative training, which minimizes the negative conditional likelihood of the query predicates given the evidence ones. Thus, in this paper we focus on this type of learning.[1]

3.1 Voted Perceptron

Gradient descent algorithms use the gradient, $\mathbf{g}$, scaled by a learning rate, η, to update the weight vector $\mathbf{w}$ in each step:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}$$

In an MLN, the derivative of the negative conditional log-likelihood (CLL) with respect to a weight is the difference of the expected number of true ground-

[1] For simplicity, we omit prior terms throughout; in our experiments, we use a zero-mean Gaussian prior on all weights with all algorithms.

ings of the corresponding clause and the actual number according to the data:

$$\frac{\partial}{\partial w_i} - \log P(Y=y|X=x) = E_w[n_i] - n_i$$

where y is the state of the non-evidence atoms in the data, and x is the state of the evidence.

The basic idea of the voted perceptron (VP) algorithm [3] is to approximate the intractable expectations $E_w[n_i]$ with the counts in the most probable explanation (MPE) state, which is the most probable state of non-evidence atoms given the evidence. To combat overfitting, instead of returning the final weights, VP returns the average of the weights from all iterations of gradient descent.

Collins originally proposed VP for training hidden Markov models discriminatively, and in this case the MPE state is unique and can be computed exactly in polynomial time using the Viterbi algorithm. In MLNs, MPE inference is intractable but can be reduced to solving a weighted maximum satisfiability problem, for which efficient algorithms exist such as MaxWalkSAT [9]. Singla and Domingos [19] use this approach and discuss how the resulting algorithm can be viewed as approximately optimizing log-likelihood. However, the use of voted perceptron in MLNs is potentially complicated by the fact that the MPE state may no longer be unique, and MaxWalkSAT is not guaranteed to find it.

3.2 Contrastive Divergence

The contrastive divergence (CD) algorithm is identical to VP, except that it approximates the expectations $E_w[n_i]$ from a small number of MCMC samples instead of using the MPE state. Using MCMC is presumably more accurate and stable, since it converges to the true expectations in the limit. While running an MCMC algorithm to convergence at each iteration of gradient descent is infeasibly slow, Hinton [8] has shown that a few iterations of MCMC yield enough information to choose a good direction for gradient descent. Hinton named this method *contrastive divergence*, because it can be interpreted as optimizing a difference of Kullback-Leibler divergences. Contrastive divergence can also be seen as an efficient way to approximately optimize log-likelihood.

The MCMC algorithm typically used with contrastive divergence is Gibbs sampling, but for MLNs the much faster alternative of MC-SAT is available. Because successive samples in MC-SAT are much less correlated than successive sweeps in Gibbs sampling, they carry more information and are likely to yield a better descent direction. In particular, the different samples are likely to be from different modes, reducing the error and potential instability associated with choosing a single mode.

In our experiments, we found that five samples were sufficient, and additional samples were not worth the time: any increased accuracy that 10 or 100 samples might bring was offset by the increased time per iteration. We avoid the need for burn-in by starting at the last state sampled in the previous iteration of gradient descent. (This differs from Hinton's approach, which always starts at the true values in the training data.)

3.3 Per-Weight Learning Rates

VP and CD are both simple gradient descent procedures, and as a result highly vulnerable to the problem of ill-conditioning. Ill-conditioning occurs when the *condition number*, the ratio between the largest and smallest absolute eigenvalues of the Hessian, is far from one. On ill-conditioned problems, gradient descent is very slow, because no single learning rate is appropriate for all weights. In MLNs, the Hessian is the negative covariance matrix of the clause counts. Because some clauses can have vastly greater numbers of true groundings than others, the variances of their counts can be correspondingly larger, and ill-conditioning becomes a serious issue.

One solution is to modify both algorithms to have a different learning rate for each weight. Since tuning every learning rate separately is impractical, we use a simple heuristic to assign a learning rate to each weight:

$$\eta_i = \frac{\eta}{n_i}$$

where η is the user-specified global learning rate and n_i is the number of true groundings of the ith formula. (To avoid dividing by zero, if $n_i = 0$ then $\eta_i = \eta$.) When computing this number, we ignore the groundings that are satisfied by the evidence (e.g., $A \Rightarrow B$ when A is false). This is because, being fixed, they cannot contribute to the variance.

We refer to the modified versions of VP and CD as VP-PW and CD-PW.

3.4 Diagonal Newton

When the function being optimized is quadratic, Newton's method can move to the global minimum or maximum in a single step. It does so by multiplying the gradient, $\mathbf{g}$, by the inverse Hessian, $\mathbf{H}^{-1}$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1}\mathbf{g}$$

When there are many weights, using the full Hessian becomes infeasible. A common approximation is to use the *diagonal* Newton (DN) method, which uses the inverse of the diagonalized Hessian in place of the inverse Hessian. DN typically uses a smaller step size than the full Newton method. This is important when applying the algorithm to non-quadratic functions, such as MLN negative CLL, where the quadratic approximation is only good within a local region.

The Hessian of the negative CLL for an MLN is simply the covariance matrix:

$$\frac{\partial}{\partial w_i \partial w_j} - \log P(Y\!=\!y|X\!=\!x) = E_w[n_i n_j] - E_w[n_i]E_w[n_j]$$

Like the gradient, this can be estimated using samples from MC-SAT. In each iteration, we take a step in the diagonalized Newton direction:

$$w_i = w_i - \alpha \, \frac{E_w[n_i] - n_i}{E_w[n_i^2] - (E_w[n_i])^2}$$

The step size α could be computed in a number of ways, including keeping it fixed, but we achieved the best results using the following method. Given a search direction $\mathbf{d}$ and Hessian matrix $\mathbf{H}$, we compute the step size as follows:

$$\alpha = \frac{-\mathbf{d}^T\mathbf{g}}{\mathbf{d}^T\mathbf{H}\mathbf{d} + \lambda\mathbf{d}^T\mathbf{d}}$$

where $\mathbf{d}$ is the search direction. For a quadratic function and $\lambda = 0$, this step size would move to the minimum function value along $\mathbf{d}$. Since our function is not quadratic, a non-zero λ term serves to limit the size of the step to a region in which our quadratic approximation is good. After each step, we adjust λ to increase or decrease the size of the so-called *model trust region* based on how well the approximation matched the function. Let Δ_{actual} be the actual change in the function value, and let Δ_{pred} be the predicted change in the function value from the previous gradient and Hessian and our last step, $\mathbf{d}_{t-1}$:

$$\Delta_{pred} = \mathbf{d}_{t-1}^T\mathbf{g}_{t-1} + 1/2\,\mathbf{d}_{t-1}^T\mathbf{H}_{t-1}\mathbf{d}_{t-1}$$

A standard method for adjusting λ is as follows [5]:

$$\begin{aligned} &\text{if } (\Delta_{actual}/\Delta_{pred} > 0.75) \;\text{ then }\; \lambda_{t+1} = \lambda_t/2 \\ &\text{if } (\Delta_{actual}/\Delta_{pred} < 0.25) \;\text{ then }\; \lambda_{t+1} = 4\lambda_t \end{aligned}$$

Since we cannot efficiently compute the actual change in negative CLL, we approximate it as the product of the step we just took and the gradient after taking it: $\Delta_{actual} = \mathbf{d}_{t-1}^T\mathbf{g}_t$. Since the negative CLL is a convex function, this product is an upper bound on the actual change. When this value is positive our CLL may be worse than before, so the step is rejected and redone after adjusting λ.

In models with thousands of weights or more, storing the entire Hessian matrix becomes impractical. However, when the Hessian appears only inside a quadratic form, as above, the value of this form can be computed simply as:

$$\mathbf{d}^T\mathbf{H}\mathbf{d} = E_w[(\textstyle\sum_i d_i n_i)^2] - (E_w[\textstyle\sum_i d_i n_i])^2$$

The product of the Hessian by a vector can also be computed compactly [14]. Note that α is computed using the full Hessian matrix, but the step direction is computed from the diagonalized approximation which is easier to invert.

Our per-weight learning rates can actually be seen as a crude approximation of the diagonal Newton method. The number of true groundings not satisfied by evidence is a heuristic approximation to the count variance, which the diagonal Newton method uses to rescale each dimension of the gradient. The diagonal Newton method, however, can adapt to changes in the second derivative at different points in the weight space. Its main limitation is that clauses can be far from uncorrelated. The next method addresses this issue.

3.5 Scaled Conjugate Gradient

Gradient descent can be sped up by, instead of taking a small step of constant size at each iteration, performing a line search to find the optimum along the chosen descent direction. However, on ill-conditioned problems this is still inefficient, because line searches along successive directions tend to partly undo the effect of each other: each line search makes the gradient along its direction zero, but the next line search will generally make it non-zero again. In long narrow valleys, instead of moving quickly to the optimum, gradient descent zigzags.

A solution to this is to impose at each step the condition that the gradient along previous directions remain zero. The directions chosen in this way are called *conjugate*, and the method *conjugate gradient* [18]. We used the Polak-Ribiere method for choosing conjugate gradients since it has generally been found to be the best-performing one. Conjugate gradient methods are some of the most efficient available, on a par with quasi-Newton ones. Unfortunately, applying them to MLNs is difficult, because line searches require computing the objective function, and therefore the partition function Z, which is highly intractable. (Computing Z is equivalent to computing all moments of the MLN, of which the gradient and Hessian are the first two.)

Fortunately, we can use the Hessian instead of a line search to choose a step size. This method is known as *scaled conjugate gradient* (SCG), and was originally proposed by Møller [12] for training neural networks. In our implementation, we choose a step size the same way as in diagonal Newton.

Conjugate gradient is usually more effective with a preconditioner, a linear transformation that attempts to reduce the condition number of the problem (e.g., [17]). Good preconditioners approximate the inverse Hessian. We use the inverse diagonal Hessian as our preconditioner. We refer to SCG with the preconditioner as PSCG.

4 Experiments

4.1 Datasets

Our experiments used two standard relational datasets representing two important relational tasks: Cora for entity resolution, and WebKB for collective classification.

The Cora dataset consists of 1295 citations of 132 different computer science papers, drawn from the Cora Computer Science Research Paper Engine. This dataset was originally labeled by Andrew McCallum[2]. We used a cleaned version from Singla and Domingos [20], with five splits for cross-validation.

The task on Cora is to predict which citations refer to the same paper, given the words in their author, title, and venue fields. The labeled data also specifies which pairs of author, title, and venue fields refer to the same entities. In our experiments, we evaluated the ability of the model to deduplicate fields as well as citations. Since the number of possible equivalances is very large, we used the canopies found by Singla and Domingos [20] to make this problem tractable.

The MLN we used for this is very similar to the "MLN(B+C+T)" model used by Singla and Domingos [20]. Its formulas link words to citation identity,

[2] http://www.cs.umass.edu/∼mccallum/data/cora-refs.tar.gz

words to field identity, and field identity to citation identity. In this way, word co-occurrence affects the probability that two citations are the same both indirectly, through field similarities, and directly. These rules are repeated for each word appearing in the database so that individualized weights can be learned, representing the relative importance of each word in each context. This model also features transitive closure for all equivalence predicates.

We did two things differently from Singla and Domingos. First, we added rules that relate words to field identity but apply equally to all words. Because these rules are not specific to particular words, they can potentially improve generalization and reduce overfitting. Secondly, we learned weights for all rules. Singla and Domingos set the weights for all word-specific rules using a naive Bayes model, and only learned the other rules' weights using VP. Our learning problem is therefore much harder and more ill-conditioned, but our more powerful algorithms enabled us to achieve the best results to date on Cora.

In our version, the total number of weights is 6141. During learning, the number of ground clauses exceeded 3 million.

The WebKB dataset consists of labeled web pages from the computer science departments of four universities. We used the relational version of the dataset from Craven and Slattery [4], which features 4165 web pages and 10,935 web links, along with the words on the webpages, anchors of the links, and neighborhoods around each link.

Each web page is marked with some subset of the categories: person, student, faculty, professor, department, research project, and course. Our goal is to predict these categories from the web pages' words and link structures.

We used a very simple MLN for this model, consisting only of formulas linking words to page classes, and page classes to the classes of linked pages. The "word-class" rules were of the following form:

$$\texttt{Has(page,word)} \Rightarrow \texttt{Class(page,class)}$$
$$\neg\texttt{Has(page,word)} \Rightarrow \texttt{Class(page,class)}$$

We learned a separate weight for each of these rules for each (word, class) pair. Classes of linked pages were related by the formula:

$$\texttt{Class(page1,class1)} \wedge \texttt{LinksTo(page1,page2)} \Rightarrow \texttt{Class(page2,class2)}$$

We learned a separate weight for this rule for each pair of classes. When instantiated for each word and class, the model contained 10,891 weights. While simple to write, this model represents a complex, non-i.i.d. probability distribution in which query predicates are linked in a large graph. During learning, the number of ground clauses exceeded 300,000.

We estimated the condition number for both Cora and WebKB at the point where all weights are zero. (Because our learning problem is not quadratic, the condition number depends on the current weights.) The size of these problems makes computing the condition number of the full Hessian matrix difficult, but we can easily compute the condition number of the diagonalized Hessian, which is simply the largest ratio of two clause variances. For Cora, this was over 600,000, while for WebKB it was approximately 7000. This indicates that both learning problems are ill-conditioned, but Cora is much worse than WebKB.

4.2 Metrics

To score our models, we ran MC-SAT for 100 burn-in and 1000 sampling iterations on the test data. The marginal conditional probability of each query atom is the fraction of samples in which the atom was true with a small prior to prevent zero counts.

From these marginal probabilities, we estimate conditional log-likelihood (CLL) by averaging the log marginal probabilities of the true values of the query predicates. CLL is the metric all of the algorithms attempt to optimize. However, in cases such as entity resolution where the class distribution is highly skewed, CLL can be a poor metric. For this reason, we also look at AUC, the area under the precision-recall curve. The disadvantage of AUC is that it ignores calibration: AUC only considers whether true atoms are given higher probability than false atoms.

4.3 Methodology

We ran our experiments using five-way cross-validation for Cora and four-way cross-validation for WebKB. For each train/test split, one of the training datasets was selected as a validation set and the remaining ones formed the tuning set. The tuning procedure consisted of training each algorithm for four hours on the tuning sets with various values of the learning rate. For each algorithm on each split, we chose the learning rates that worked best on the corresponding validation set for each evaluation metric.

We used the implementation of voted perceptron for MLNs in the Alchemy package [11], and implemented the other algorithms as extensions of Alchemy. For DN, SCG, and PSCG, we started with $\lambda = 1$ and let the algorithm adjust it automatically. For algorithms based on MC-SAT, we used 5 samples of MC-SAT for each iteration of the learning algorithm. The width of the Gaussian prior was set for each dataset based on preliminary experiments.

After tuning all algorithms, we reran them for 10 hours with their respective training sets, including the held-out validation data. For the gradient descent algorithms, we averaged the weights from all iterations.

4.4 Results

Our results for the Cora and WebKB datasets are shown in Figure 1. Error bars are omitted for clarity; at the final data point, all differences exceed twice the standard error. For AUC, we computed the standard deviation using the technique of Richardson and Domingos [16].

PSCG is the most accurate of all the algorithms compared, obtaining the best CLL and AUC on both Cora and WebKB. It converges relatively quickly as

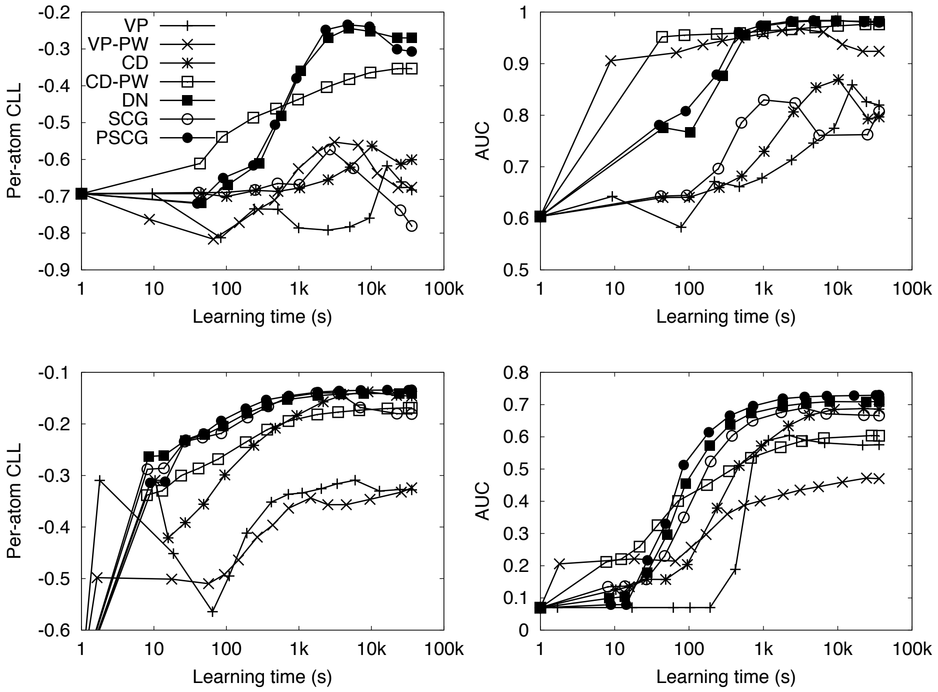

Fig. 1. CLL and AUC for Cora (above) and WebKB (below). Learning times are shown on a logarithmic scale.

well: on WebKB, the PSCG learning curve dominates all others after 2 minutes; on Cora, it dominates after 15 minutes. DN is consistently close behind PSCG in CLL and AUC, briefly doing better when PSCG starts to overfit. In contrast, VP and CD consistently converge more slowly to worse AUC and CLL.

On Cora, the algorithms that adjust the search direction using true clause counts or count variance do much better than those that do not. This suggests that these techniques help greatly in cases of extreme ill-conditioning. Without a preconditioner, even SCG does poorly. This is because, like VP and CD, the first step it takes is in the direction of the gradient. On a very ill-conditioned dataset like Cora, the gradient is a very poor choice of search direction.

The AUC results we show for Cora are for all query predicates—`SameAuthor`, `SameVenue`, `SameTitle`, and `SameBib`. When computing the AUC for just the `SameBib` predicate, PSCG reaches a high of 0.992 but ends at 0.990 after overfitting slightly. DN and CD-PW do about the same, ending at AUCs of 0.992 and 0.991, respectively. All of these algorithms exceed the 0.988 AUC reported by Singla and Domingos [20], the best previously published result on this dataset, and they do so by more than twice the standard error.

On WebKB, the ill-conditioning is less of an issue. PSCG still does better than SCG, but not drastically better. VP-PW and CD-PW actually do worse than VP and CD. This is because the per-weight learning rates are much smaller for the relational rules than the word-specific rules. This makes the relational rules converge much more slowly than they should.

The performance of some of the algorithms sometimes degrades with additional learning time. For some of the algorithms, such as PSCG, DN, and VP-PW on Cora, this is simply a symptom of overfitting. More careful tuning or a better prior could help correct this. But for other algorithms, such as SCG and VP on Cora, the later models perform worse on training data as well. For SCG, this seems to be the result of noisy inference and very ill-conditioned problems, which cause even a slight error in the step direction to potentially have a significant effect. Our lower bound on the improvement in log-likelihood prevents this in theory, but in practice a noisy gradient may still cause us to take bad steps. PSCG suffers much less from this effect, since the preconditioning makes the learning problem better behaved. For VP and CD, the most likely cause is learning rates that are too high. Our tuning experiments selected the learning rates that worked best after four hours on a smaller set of data. The increased amount of data in the test scenario increased the magnitude of the gradients, making these learning rates less stable than they were in the tuning scenario. This extreme sensitivity to learning rate makes learning good models with VP and CD much more difficult. We also experimented with the stochastic meta-descent algorithm [21], which automatically adjusts learning rates in each dimension, but found it to be too unstable for these domains.

In sum, the MLN weight learning methods we have introduced in this paper greatly outperform the voted perceptron. Given similar learning time, they learn much more accurate models; and, judging from the curves in Figure 1, running VP until it reaches the same accuracy as the better algorithms would take an extremely long time.

5 Conclusion

Weight learning for Markov logic networks can be extremely ill-conditioned, making simple gradient descent-style algorithms very slow to converge. In this paper we studied a number of more sophisticated alternatives, of which the best-performing one is preconditioned scaled conjugate gradient. This can be attributed to its effective use of second-order information. However, the simple heuristic of dividing the learning rate by the true clause counts for each weight can sometimes give very good results. Using one of these methods instead of gradient descent can yield a much better model in less time.

Acknowledgments. This research was funded by a Microsoft Research fellowship awarded to the first author, DARPA contract NBCH-D030010/02-000225, DARPA grant FA8750-05-2-0283, NSF grant IIS-0534881, and ONR grant N-00014-05-1-0313. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, NSF, ONR, or the United States Government. Thanks to Hoifung Poon for formula corrections.

References

1. S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second order methods. In *Proc. 1988 Connectionist Models Summer School*, pages 29–37, 1989. Morgan Kaufmann.
2. J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B*, 48:259–302, 1986.
3. M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. CEMNLP-2002*, 2002.
4. M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1/2):97–119, 2001.
5. R. Fletcher. *Practical Methods of Optimization.* Wiley-Interscience, New York, NY, second edition, 1987.
6. L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning.* MIT Press, 2007.
7. W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice.* Chapman and Hall, London, UK, 1996.
8. G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.
9. H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Du, J. Gu, and P. M. Pardalos, editors, *The Satisfiability Problem: Theory and Applications*, pages 573–586. American Mathematical Society, New York, NY, 1996.
10. S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proc. ICML-2005*, pages 441–448, 2005. ACM Press.
11. S. Kok, P. Singla, M. Richardson, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2005. http://alchemy.cs.washington.edu/.
12. M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.
13. J. Nocedal and S. Wright. *Numerical Optimization.* Springer, New York, NY, 2006.
14. B. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.
15. H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proc. AAAI-2006*, pages 458–463, 2006. AAAI Press.
16. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.
17. F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. ACL-2003*, 2003.
18. J. Shewchuck. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.
19. P. Singla and P. Domingos. Discriminative training of Markov logic networks. In *Proc. AAAI-2005*, pages 868–873, 2005. AAAI Press.
20. P. Singla and P. Domingos. Entity resolution with Markov logic. In *Proc. ICDM-2006*, pages 572–582, 2006. IEEE Computer Society Press.
21. S. Vishwanathan, N. Schraudolph, M. Schmidt, and K. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *Proc. ICML-2006*, 2006.