Efficient Inference of CRFs for Large-Scale Natural Language Data

Minwoo Jeong[†*] **Chin-Yew Lin**[‡] **Gary Geunbae Lee**[†]
[†]Pohang University of Science & Technology, Pohang, Korea
[‡]Microsoft Research Asia, Beijing, China
[†]{stardust,gblee}@postech.ac.kr [‡]cyl@microsoft.com

Abstract

This paper presents an efficient inference algorithm of conditional random fields (CRFs) for large-scale data. Our key idea is to decompose the output label state into an active set and an inactive set in which most unsupported transitions become a constant. Our method unifies two previous methods for efficient inference of CRFs, and also derives a simple but robust special case that performs faster than exact inference when the active sets are sufficiently small. We demonstrate that our method achieves dramatic speedup on six standard natural language processing problems.

1 Introduction

Conditional random fields (CRFs) are widely used in natural language processing, but extending them to large-scale problems remains a significant challenge. For simple graphical structures (e.g. linear-chain), an exact inference can be obtained efficiently if the number of output labels is not large. However, for large number of output labels, the inference is often prohibitively expensive.

To alleviate this problem, researchers have begun to study the methods of increasing inference speeds of CRFs. Pal et al. (2006) proposed a Sparse Forward-Backward (SFB) algorithm, in which marginal distribution is compressed by approximating the true marginals using Kullback-Leibler (KL) divergence. Cohn (2006) proposed a Tied Potential (TP) algorithm which constrains the labeling considered in each feature function, such that the functions can detect only a relatively small set of labels. Both of these techniques efficiently compute the marginals with a significantly reduced runtime, resulting in faster training and decoding of CRFs.

This paper presents an efficient inference algorithm of CRFs which unifies the SFB and TP approaches. We first decompose output labels states into *active* and *inactive* sets. Then, the active set is selected by feasible heuristics and the parameters of the inactive set are held a constant. The idea behind our method is that not all of the states contribute to the marginals, that is, only a small group of the labeling states has sufficient statistics. We show that the SFB and the TP are special cases of our method because they derive from our unified algorithm with a different setting of parameters. We also present a simple but robust variant algorithm in which CRFs efficiently learn and predict large-scale natural language data.

[*]Parts of this work were conducted during the author's internship at Microsoft Research Asia.

2 Linear-chain CRFs

Many versions of CRFs have been developed for use in natural language processing, computer vision, and machine learning. For simplicity, we concentrate on linear-chain CRFs (Lafferty et al., 2001; Sutton and McCallum, 2006), but the generic idea described here can be extended to CRFs of any structure.

Linear-chain CRFs are conditional probability distributions over *label sequences* which are conditioned on *input sequences* (Lafferty et al., 2001). Formally, $\mathbf{x} = \{x_t\}_{t=1}^{T}$ and $\mathbf{y} = \{y_t\}_{t=1}^{T}$ are sequences of input and output variables. Respectively, where T is the length of sequence, $x_t \in \mathcal{X}$ and $y_t \in \mathcal{Y}$ where $\mathcal{X}$ is the finite set of the input observations and $\mathcal{Y}$ is that of the *output label* state space. Then, a first-order linear-chain CRF is defined as:

$$p_{\boldsymbol{\lambda}}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}), \qquad (1)$$

where Ψ_t is the local potential that denotes the factor at time t, and $\boldsymbol{\lambda}$ is the parameter vector. $Z(\mathbf{x})$ is a partition function which ensures the probabilities of all state sequences sum to one. We assume that the potentials factorize according to a set of observation features $\{\phi_k^1\}$ and transition features $\{\phi_k^2\}$, as follows:

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}) = \Psi_t^1(y_t, \mathbf{x}) \cdot \Psi_t^2(y_t, y_{t-1}), \qquad (2)$$

$$\Psi_t^1(y_t, \mathbf{x}) = e^{\sum_k \lambda_k^1 \phi_k^1(y_t, \mathbf{x})}, \qquad (3)$$

$$\Psi_t^2(y_t, y_{t-1}) = e^{\sum_k \lambda_k^2 \phi_k^2(y_t, y_{t-1})}, \qquad (4)$$

where $\{\lambda_k^1\}$ and $\{\lambda_k^2\}$ are weight parameters which we wish to learn from data.

Inference is significantly challenging both in learning and decoding CRFs. Time complexity is $\mathcal{O}(T|\mathcal{Y}|^2)$ for exact inference (i.e., forward-backward and Viterbi algorithm) of linear-chain CRFs (Lafferty et al., 2001). The inference process is often prohibitively expensive

Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, pages 281–284,
Suntec, Singapore, 4 August 2009.

when $|\mathcal{Y}|$ is large, as is common in large-scale tasks. This problem can be alleviated by introducing approximate inference methods based on reduction of the search spaces to be explored.

3 Efficient Inference Algorithm

3.1 Method

The key idea of our proposed efficient inference method is that the output label state $\mathcal{Y}$ can be decomposed to an *active* set $\mathcal{A}$ and an *inactive* set $\mathcal{A}^c$. Intuitively, many of the possible transitions $(y_{t-1} \rightarrow y_t)$ do not occur, or are unsupported, that is, only a small part of the possible labeling set is informative. The inference algorithm need not precisely calculate marginals or maximums (more generally, messages) for unsupported transitions. Our efficient inference algorithm approximates the unsupported transitions by assigning them a constant value. When $|\mathcal{A}| < |\mathcal{Y}|$, both training and decoding times are remarkably reduced by this approach.

We first define the notation for our algorithm. Let $\mathcal{A}_i$ be the active set and $\mathcal{A}_i^c$ be the inactive set of output label i where $\mathcal{Y}_i = \mathcal{A}_i \cup \mathcal{A}_i^c$. We define $\mathcal{A}_i$ as:

$$\mathcal{A}_i = \{j | \delta(y_t = i, y_{t-1} = j) > \epsilon\} \tag{5}$$

where δ is a criterion function of transitions $(y_{t-1} \rightarrow y_t)$ and ϵ is a hyperparameter. For clarity, we define the local factors as:

$$\Psi^1_{t,i} \triangleq \Psi^1_t(y_t = i, \mathbf{x}), \tag{6}$$

$$\Psi^2_{j,i} \triangleq \Psi^2_t(y_{t-1} = j, y_t = i). \tag{7}$$

Note that we can ignore the subscript t at $\Psi^2_t(y_{t-1} = j, y_t = i)$ by defining an HMM-like model, that is, transition matrix $\Psi^2_{j,i}$ is independent of t.

As exact inference, we use the forward-backward procedure to calculate marginals (Sutton and McCallum, 2006). We formally describe here an efficient calculation of α and β recursions for the forward-backward procedure. The forward value $\alpha_t(i)$ is the sum of the unnormalized scores for all partial paths that start at $t = 0$ and converge at $y_t = i$ at time t. The backward value $\beta_t(i)$ similarly defines the sum of unnormalized scores for all partial paths that start at time $t + 1$ with state $y_{t+1} = j$ and continue until the end of the sequences, $t = T + 1$. Then, we decompose the equations of exact α and β recursions as follows:

$$\alpha_t(i) = \Psi^1_{t,i} \left(\sum_{j \in \mathcal{A}_i} \Big(\Psi^2_{j,i} - \omega\Big) \alpha_{t-1}(j) + \omega \right), \tag{8}$$

$$\beta_{t-1}(j) = \sum_{i \in \mathcal{A}_j} \Psi^1_{t,i} \Big(\Psi^2_{j,i} - \omega\Big) \beta_t(i) + \omega \sum_{i \in \mathcal{Y}} \Psi^1_{t,i} \beta_t(i), \tag{9}$$

where ω is a shared transition parameter value for set $\mathcal{A}_i^c$, that is, $\Psi^2_{j,i} = \omega$ if $j \in \mathcal{A}_i^c$. Note that $\sum_i \alpha_t(i) = 1$ (Sutton and McCallum, 2006). Because all unsupported transitions in $\mathcal{A}_i^c$ are calculated simultaneously, the complexities of Eq. (8) and (9) are approximately $\mathcal{O}(T|\mathcal{A}_{avg}||\mathcal{Y}|)$ where $|\mathcal{A}_{avg}|$ is the average number of states in the active set, i.e., $\frac{1}{T}\sum_{t=1}^{T} |\mathcal{A}_i|$. The worst case complexity of our α and β equations is $\mathcal{O}(T|\mathcal{Y}|^2)$.

Similarly, we decompose a γ recursion for the Viterbi algorithm as follows:

$$\gamma_t(i) = \Psi^1_{t,i} \left\{ \max \left(\max_{j \in \mathcal{A}_i} \Psi^2_{j,i} \gamma_{t-1}(j), \max_{j \in \mathcal{Y}} \omega \gamma_{t-1}(j) \right) \right\}, \tag{10}$$

where $\gamma_t(i)$ is the sum of unnormalized scores for the best-scored partial path that starts at time $t = 0$ and converges at $y_t = i$ at time t. Because ω is constant, $\max_{j \in \mathcal{Y}} \gamma_{t-1}(j)$ can be pre-calculated at time $t - 1$. By analogy with Eq. (8) and (9), the complexity is approximately $\mathcal{O}(T|\mathcal{A}_{avg}||\mathcal{Y}|)$.

3.2 Setting δ and ω

To implement our inference algorithm, we need a method of choosing appropriate values for the setting function δ of the active set and for the constant value ω of the inactive set. These two problems are closely related. The size of the active set affects both the complexity of inference algorithm and the quality of the model. Therefore, our goal for selecting δ and ω is to make a plausible assumption that does not sacrifice much accuracy but speeds up when applying large state tasks. We describe four variant special case algorithms.

Method 1: We set $\delta(i, j) = Z(L)$ and $\omega = 0$ where L is a beam set, $L = \{l_1, l_2, \ldots, l_m\}$ and the sub-partition function $Z(L)$ is approximated by $Z(L) \approx \alpha_{t-1}(j)$. In this method, all sub-marginals in the inactive set are totally excluded from calculation of the current marginal. α and β in the inactive sets are set to 0 by default. Therefore, at each time step t the algorithm prunes all states i in which $\alpha_t(i) < \epsilon$. It also generates a subset L of output labels that will be exploited in next time step $t + 1$.[1] This method has been derived theoretically from the process of selecting a compressed marginal distribution within a fixed KL divergence of the true marginal (Pal et al., 2006). This method most closely resembles SFB algorithm; hence we refer an alternative of SFB.

Method 2: We define $\delta(i, j) = |\Psi^2_{j,i} - 1|$ and $\omega = 1$. In practice, unsupported transition features are not parameterized[2]; this means that $\lambda_k = 0$ and $\Psi^2_{j,i} = 1$ if $j \in \mathcal{A}_i^c$. Thus, this method estimates nearly-exact

[1] In practice, dynamically selecting L increases the number of computations, and this is the main disadvantage of Method 1. However, in inactive sets $\alpha_{t-1}(j) = 0$ by default; hence, we need not calculate $\beta_{t-1}(j)$. Therefore, it counterbalances the extra computations in β recursion.

[2] This is a common practice in implementation of input and output joint feature functions for large-scale problems. This scheme uses only supported features that are used at least once in the training examples. We call it the sparse model. While a complete and dense feature model may per-

CRFs if the hyperparameter is $\epsilon = 0$; hence this criterion does not change the parameter. Although this method is simple, it is sufficiently efficient for training and decoding CRFs in real data.

Method 3: We define $\delta(i,j) = E_{\tilde{p}}\langle\phi_k^2(i,j)\rangle$ where $E_{\tilde{p}}\langle z\rangle$ is an empirical count of event z in training data. We also assign a real value for the inactive set, i.e., $\omega = c \in \mathbb{R}, c \neq 0, 1$. The value c is estimated in the training phase; hence, c is a shared parameter for the inactive set. This method is equivalent to TP (Cohn, 2006). By setting ϵ larger, we can achieve faster inference, a tradeoff exists between efficiency and accuracy.

Method 4: We define the shared parameter as a function of output label y in the inactive set, i.e., $c(y)$. As in Method 3, $c(y)$ is estimated during the training phase. When the problem expects different aspects of unsupported transitions, this method would be better than using only one parameter c for all labels in inactive set.

4 Experiment

We evaluated our method on six large-scale natural language data sets (Table 1): Penn Treebank[3] for part-of-speech tagging (PTB), phrase chunking data[4] (CoNLL00), named entity recognition data[5] (CoNLL03), grapheme-to-phoneme conversion data[6] (NetTalk), spoken language understanding data (Communicator) (Jeong and Lee, 2006), and fine-grained named entity recognition data (Encyclopedia) (Lee et al., 2007). The active set is sufficiently small in Communicator and Encyclopedia despite their large numbers of output labels. In all data sets, we selected the current word, ± 2 context words, bigrams, trigrams, and prefix and suffix features as basic feature templates. A template of part-of-speech tag features was added for CoNLL00, CoNLL03, and Encyclopedia. In particular, all tasks except PTB and NetTalk require assigning a label to a phrase rather than to a word; hence, we used standard "BIO" encoding. We used un-normalized log-likelihood, accuracy and training/decoding times as our evaluation measures. We did not use cross validation and development set for tuning the parameter because our goal is to evaluate the efficiency of inference algorithms. Moreover, using the previous state-of-the-art features we expect the achievement of better accuracy.

All our models were trained until parameter estimation converged with a Gaussian prior variance of 4. During training, a pseudo-likelihood parameter estimation (Sutton and McCallum, 2006) was used as an initial weight (estimated in 30 iterations). We used complete and dense input/output joint features for dense model (Dense), and only supported features that are used at least once in the training examples for sparse model (Sparse). All of our model variants were based on Sparse model. For the hyper parameter ϵ, we empirically selected 0.001 for Method 1 (this preserves 99% of probability density), 0 for Method 2, and 4 for Methods 3 and 4. Note that ϵ for Methods 2, 3, and 4 indicates an empirical count of features in training set. All experiments were implemented in C++ and executed in Windows 2003 with XEON 2.33 GHz Quad-Core processor and 8.0 Gbyte of main memory.

Table 1: Data sets: number of sentences in the training (#Train) and the test data sets (#Test), and number of output labels (#Label). $|\mathcal{A}_{avg}^{\omega=1}|$ denotes the average number of active set when $\omega = 1$, i.e., the supported transitions that are used at least once in the training set.

Set	#Train	#Test	#Label	$\lvert\mathcal{A}_{avg}^{\omega=1}\rvert$
PTB	38,219	5462	45	30.01
CoNLL00	8,936	2,012	22	6.59
CoNLL03	14,987	3,684	8	4.13
NetTalk	18,008	2,000	51	22.18
Communicator	13,111	1,193	120	3.67
Encyclopedia	25,348	6,336	279	3.27

We first show that our method is efficient for learning CRFs (Figure 1). In all learning curves, Dense generally has a higher training log-likelihood than Sparse. For PTB and Encyclopedia, results for Dense are not available because training in a single machine failed due to out-of-memory errors. For both Dense and Sparse, we executed the exact inference method. Our proposed method (Method 1$\sim$4) performs faster than Sparse. In most results, Method 1 was the fastest, because it was terminated after fewer iterations. However, Method 1 sometimes failed to converge, for example, in Encyclopedia. Similarly, Method 3 and 4 could not find the optimal solution in the NetTalk data set. Method 2 showed stable results.

Second, we evaluated the accuracy and decoding time of our methods (Table 2). Most results obtained using our method were as accurate as those of Dense and Sparse. However, some results of Method 1, 3, and 4 were significantly inferior to those of Dense and Sparse for one of two reasons: 1) parameter estimation failed (NetTalk and Encyclopedia), or 2) approximate inference caused search errors (CoNLL00 and Communicator). The improvements of decoding time on Communicator and Encyclopedia were remarkable.

Finally, we compared our method with two open-source implementations of CRFs: MALLET[7] and CRF++[8]. MALLET can support the Sparse model, and the CRF++ toolkit implements only the Dense model. We compared them with Method 2 on the Communicator data set. In the accuracy measure, the results were 91.56 (MALLET), 91.87 (CRF++), and 91.92 (ours). Our method performs 5$\sim$50 times faster for training (1,774 s for MALLET, 18,134 s for CRF++,

form better, the sparse model performs well in practice without significant loss of accuracy (Sha and Pereira, 2003).

[3]Penn Treebank3: Catalog No. LDC99T42

[4]http://www.cnts.ua.ac.be/conll2000/chunking/

[5]http://www.cnts.ua.ac.be/conll2003/ner/

[6]http://archive.ics.uci.edu/ml/

[7]Ver. 2.0 RC3, http://mallet.cs.umass.edu/

[8]Ver. 0.51, http://crfpp.sourceforge.net/

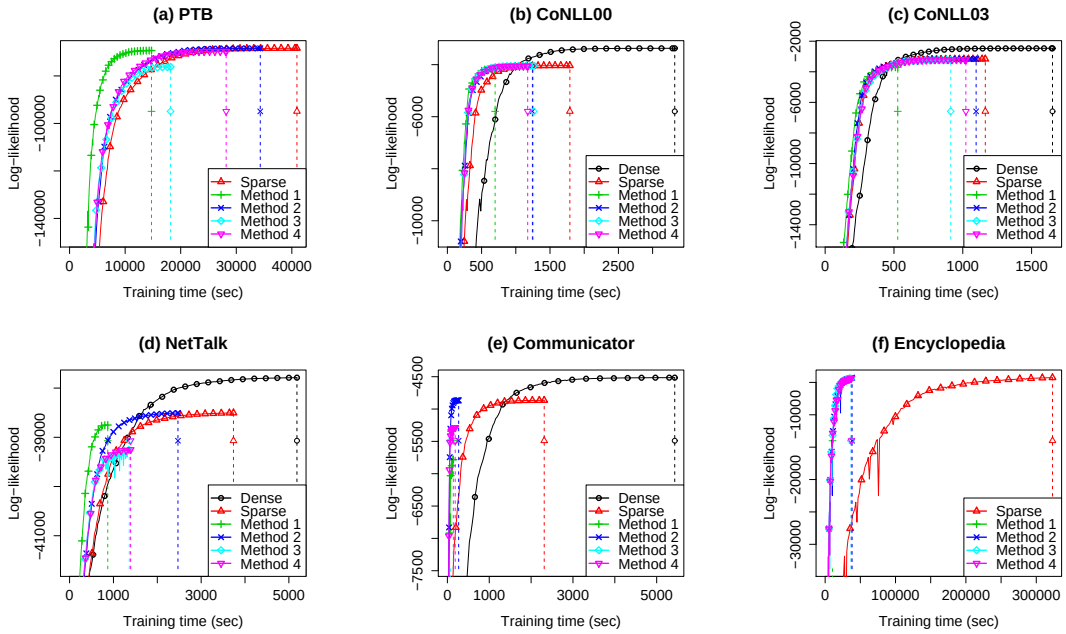

Figure 1: Result of training linear-chain CRFs: Un-normalized training log-likelihood and training times are compared. Dashed lines denote the termination of training step.

Table 2: Decoding result; columns are percent accuracy (Acc), and decoding time in milliseconds (Time) measured per testing example. '*' indicates that the result is significantly different from the Sparse model. N/A indicates failure due to out-of-memory error.

Method	PTB		CoNLL00		CoNLL03		NetTalk		Communicator		Encyclopedia	
	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time
Dense	N/A	N/A	96.1	0.89	95.8	0.26	88.4	0.49	91.6	0.94	N/A	N/A
Sparse	96.6	1.12	95.9	0.62	95.9	0.21	88.4	0.44	91.9	0.83	93.6	34.75
Method 1	96.8	0.74	95.9	0.55	*94.0	0.24	*88.3	0.34	91.7	0.73	*69.2	15.77
Method 2	96.6	0.92	*95.7	0.52	95.9	0.21	*87.4	0.32	91.9	0.30	93.6	4.99
Method 3	96.5	0.84	*94.2	0.51	95.9	0.24	*78.2	0.29	*86.7	0.30	93.7	6.14
Method 4	96.6	0.85	*92.1	0.51	95.9	0.24	*77.9	0.30	91.9	0.29	93.3	4.88

and 368 s for ours) and 7∼12 times faster for decoding (2.881 ms for MALLET, 5.028 ms for CRF++, and 0.418 ms for ours). This result demonstrates that learning and decoding CRFs for large-scale natural language problems can be efficiently solved using our method.

5 Conclusion

We have demonstrated empirically that our efficient inference method can function successfully, allowing for a significant speedup of computation. Our method links two previous algorithms, the SFB and the TP. We have also showed that a simple and robust variant method (Method 2) is effective in large-scale problems.[9] The empirical results show a significant improvement in the training and decoding speeds especially when the problem has a large state space of output labels. Future work will consider applications to other large-scale problems, and more-general graph topologies.

[9]Code used in this work is available at http://argmax.sourceforge.net/.

References

T. Cohn. 2006. Efficient inference in large conditional random fields. In *Proc. ECML*, pages 606–613.

M. Jeong and G. G. Lee. 2006. Exploiting non-local features for spoken language understanding. In *Proc. of COLING/ACL*, pages 412–419, Sydney, Australia, July.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, pages 282–289.

C. Lee, Y. Hwang, and M. Jang. 2007. Fine-grained named entity recognition and relation extraction for question answering. In *Proc. SIGIR Poster*, pages 799–800.

C. Pal, C. Sutton, and A. McCallum. 2006. Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *Proc. ICASSP*.

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proc. of NAACL/HLT*, pages 134–141.

C. Sutton and A. McCallum. 2006. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA.