High-Performance Training of Conditional Random Fields for Large-Scale Applications of Labeling Sequence Data

Xuan-Hieu PHAN†, Le-Minh NGUYEN†, Nonmembers, Yasushi INOGUCHI†, and Susumu HORIGUCHI††, Members

SUMMARY Conditional random fields (CRFs) have been successfully applied to various applications of predicting and labeling structured data, such as natural language tagging & parsing, image segmentation & object recognition, and protein secondary structure prediction. The key advantages of CRFs are the ability to encode a variety of overlapping, non-independent features from empirical data as well as the capability of reaching the global normalization and optimization. However, estimating parameters for CRFs is very time-consuming due to an intensive forward-backward computation needed to estimate the likelihood function and its gradient during training. This paper presents a high-performance training of CRFs on massively parallel processing systems that allows us to handle huge datasets with hundreds of thousand data sequences and millions of features. We performed the experiments on an important natural language processing task (text chunking) on large-scale corpora and achieved significant results in terms of both the reduction of computational time and the improvement of prediction accuracy.

key words: parallel computing, probabilistic graphical models, conditional random fields, structured prediction, text processing

1. Introduction

CRF, a conditionally trained Markov random field model, together with its variants have been successfully applied to various applications of predicting and labeling structured data, such as information extraction [1], [2], natural language tagging & parsing [3], [4], pattern recognition & computer vision [5]–[8], and protein secondary structure prediction [9], [10]. The key advantages of CRFs are the ability to encode a variety of overlapping, non-independent features from empirical data as well as the capability of reaching the global normalization and optimization.

However, training CRFs, i.e., estimating parameters for CRF models, is very expensive due to a heavy forward-backward computation needed to estimate the likelihood function and its gradient during the training process. The computational time of CRFs is even larger when they are trained on large-scale datasets or using higher-order Markov dependencies among states. Thus, most previous work either evaluated CRFs on moderate datasets or used the first-order Markov CRFs (i.e., the simplest configuration in which the current state only depends on one previous state). Obviously, this difficulty prevents us to explore the limit of the prediction power of high-order Markov CRFs as well as to deal with large-scale structured prediction problems.

In this paper, we present a high-performance training of CRFs on massively parallel processing systems that allows to handle huge datasets with hundreds of thousand data sequences and millions of features. Our major motivation behind this work is threefold:

• Today, (semi-)structured data (e.g., text, image, video, protein sequences) can be easily gathered from different sources, such as online documents, sensors, cameras, and biological experiments & medical tests. Thus, the need for analyzing, e.g., segmentation and prediction, those kinds of data is increasing rapidly. Building high-performance prediction models on distributed processing systems is an appropriate strategy to deal with such huge real-world datasets.

• CRF has been known as a powerful probabilistic graphical model, and already applied successfully to many learning tasks. However, there is no thoroughly empirical study on this model on large datasets to confirm its actual limit of learning capability. Our work also aims at exploring this limit in the viewpoint of empirical evaluation.

• Also, we expect to examine the extent to which CRFs with the global normalization and optimization could do better than other classifiers when performing structured prediction on large-scale datasets. And from that we want to determine whether or not the prediction accuracy of CRFs should compensate its large computational cost.

The rest of the paper is organized as follows. Section 2 briefly presents the related work. Section 3 gives the background of CRFs. Section 4 presents the parallel training of CRFs. Section 5 presents the empirical evaluation. And some conclusions are given in Section 6.

2. Related Work

Most previous researches evaluated CRFs on moderate datasets. One of the most typical and successful applications of CRFs is text shallow parsing [4]. The authors used the second-order Markov CRFs and reported the state-of-the-art accuracy for noun phrase chunking on the CoNLL2000 shared task. However, their train-
The task of predicting a label sequence to an observation sequence. For example, consider the natural language processing task of predicting the part-of-speech (POS) tag sequence. Consider the sentence: “Cars in 1990. expects its U.S. sales to remain steady at about 1,200 cars in 1990.”

Our work is also closely related to advanced optimization methods because the training of CRFs, ultimately, can be seen as an unconstrained convex optimization task. To support high-performance optimization, TAO (Toolkit for Advanced Optimization) [11] provides a convenient framework that allows users to perform large-scale optimization problems on massively parallel computers quite easily. In principle, our system can be built upon TAO framework. However, to perform many other operations other than optimization, we decided to develop our own system from scratch to keep it portable and easy to use.

3. Conditional Random Fields

The task of predicting a label sequence to an observation sequence arises in many fields, including bioinformatics, computational linguistics, and speech recognition. For example, consider the natural language processing task of predicting the part-of-speech (POS) tag sequence for an input text sequence as follows:

- Input sentence: “Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars in 1990.”
- Output sentence and POS tags: “Rolls-Royce NNPS NNP Cars NNPS NNP Inc. NNP said VBD at PRP expects VBZ its PRP$ U.S. NNP sales NNPS to TO remain VB steady JJ at IN about IN 1,200 CD cars NNS in IN 1990 CD ...”

Here, “Rolls-Royce Motor Cars Inc. said . . .” and “NNP NNP NNP NNP VBD ...” can be seen as the input data observation sequence and the output label sequence, respectively. The problem of labeling sequence data is to predict the most likely label sequence of an input data observation sequence. CRFs [12] were deliberately designed to deal with such kind of problem.

Let \( \mathbf{o} = (o_1, \ldots, o_T) \) be some input data observation sequence. Let \( \mathcal{S} \) be a finite set of states, each associated with a label \( l \in \mathcal{L} = \{ l_1, \ldots, l_Q \} \). Let \( \mathbf{s} = (s_1, \ldots, s_T) \) be some state sequence. CRFs are defined as the conditional probability of a state sequence given an observation sequence as,

\[
p_o(s|\mathbf{o}) = \frac{1}{Z(\mathbf{o})} \exp \left( \sum_{t=1}^{T} F(s, \mathbf{o}, t) \right),
\]

where \( Z(\mathbf{o}) = \sum_{s} \exp \left( \sum_{t=1}^{T} F(s^t, \mathbf{o}) \right) \) is a normalized factor summing over all label sequences. \( F(s, \mathbf{o}, t) \) is the sum of CRF features at time position \( t \),

\[
F(s, \mathbf{o}, t) = \sum_{i} \lambda_i f_i(s_{t-1}, s_t) + \sum_{j} \lambda_j g_j(o, s_t)
\]

where \( f_i \) and \( g_j \) are edge and state features, respectively; \( \lambda_i \) and \( \lambda_j \) are the feature weights associated with \( f_i \) and \( f_j \). Edge and state features are defined as binary functions as follows,

\[
\begin{align*}
\lambda_i &= \begin{cases} 1 & \text{if } x_i(o, t) = x_{i}'(t) \text{ and } s_i = l_i \\ 0 & \text{otherwise} \end{cases} \\
\lambda_j &= \begin{cases} 1 & \text{if } s_{t-1} = l_i \text{ and } x_j(o, t) = x_{j}'(t) \\ 0 & \text{otherwise} \end{cases}
\end{align*}
\]

where \( l_i \) equals 1 if the label associated with state \( s_i \) is \( l_i \), and 0 otherwise (the same for \( s_i = l_i' \)).

\[x_i(o, t)\] is a logical context predicate that indicates whether the observation sequence \( o \) (at time \( t \)) holds a particular property. \( x_j(o, t) \) is equal to 1 if \( x_j(o, t) \) is true, and 0 otherwise. Intuitively, an edge feature encodes a sequential dependency or causal relationship between two consecutive states, e.g., “the label of the previous word is JJ (adjective) and the label of the current word is NN (noun)”.

3.1 Inference in Conditional Random Fields

Inference in CRFs is to find the most likely state sequence \( s^* \) given the input observation sequence \( o \),

\[
s^* = \arg \max_s p_o(s|o) = \arg \max_s \left\{ \exp \left( \sum_{t=1}^{T} F(s, o, t) \right) \right\}
\]

In order to find \( s^* \), one can apply a dynamic programming technique with a slightly modified version of the original Viterbi algorithm for HMMs [13]. To avoid an exponential–time search over all possible settings of...
s. Viterbi stores the probability of the most likely path up to time \( t \) which accounts for the first \( t \) observations and ends in state \( s_t \). We denote this probability to be \( \varphi_t(s_t) \) (\( 0 \leq t \leq T - 1 \)) and \( \varphi_0(s_0) \) to be the probability of starting in each state \( s_0 \). The recursion is given by:

\[
\varphi_{t+1}(s_{t+1}) = \max_{s_t} \{ \varphi_t(s_t) \exp \mathbf{F}(s, o, t + 1) \} \tag{4}
\]

The recursion stops when \( t = T - 1 \) and the biggest unnormalized probability is \( p_\theta^* = \arg \max_t [\varphi_T(s_t)] \). At this time, we can backtrack through the stored information to find the most likely sequence \( s^* \).

### 3.2 Training Conditional Random Fields

CRFs are trained by setting the set of weights \( \theta = \{ \lambda_1, \lambda_2, \ldots \} \) to maximize the log-likelihood, \( L \), of a given training data set \( \mathcal{D} = \{(o^{(j)}, y^{(j)})\}_{j=1}^N \):

\[
L = \sum_{j=1}^N \log \left( p_\theta(y^{(j)}|o^{(j)}) \right) = \sum_{j=1}^N \sum_{t=1}^T F(y^{(j)}, o^{(j)}, t) - \sum_{j=1}^N \log Z(o^{(j)}) \tag{5}
\]

When the labels make the state sequence unambiguous, the likelihood function in exponential models such as CRFs is convex, thus searching the global optimum is guaranteed. However, the convex function can not be found analytically. Parameter estimation for CRFs requires an iterative procedure. It has been shown that quasi-Newton methods, such as L-BFGS [14], are most efficient [4]. This method can avoid the explicit estimation of the Hessian matrix of the log-likelihood by building up an approximation of it using successive evaluations of the gradient. L-BFGS is a limited-memory quasi-Newton procedure for unconstrained convex optimization that requires the value and gradient vector of the function to be optimized. The log-likelihood gradient component of \( \lambda_k \) is

\[
\frac{\delta L}{\lambda_k} = \sum_{j=1}^N \left[ \tilde{c}_k(y^{(j)}, o^{(j)}) - \sum_s p_\theta(s|o^{(j)})C_k(s, o^{(j)}) \right] = \sum_{j=1}^N \left[ \tilde{c}_k(y^{(j)}, o^{(j)}) - \mathbb{E}_{p_\theta} C_k(s, o^{(j)}) \right] \tag{6}
\]

where \( \tilde{c}_k(y^{(j)}, o^{(j)}) = \sum_{t=1}^T f_k(y^{(j)}, o^{(j)}, t) \) if \( \lambda_k \) is associated with an edge feature \( f_k \) and = \( \sum_{t=1}^T g_k(o^{(j)}, y^{(j)}, t) \) if \( \lambda_k \) is associated with a state feature \( g_k \). Intuitively, it is the expectation (i.e., the count) of feature \( f_k \) (or \( g_k \)) with respect to the \( j \)th training sequence of the empirical data \( \mathcal{D} \). And \( \mathbb{E}_{p_\theta} C_k(s, o^{(j)}) \) is the expectation (i.e., the count) of feature \( f_k \) (or \( g_k \)) with respect to the CRF model \( p_\theta \).

The training process for CRFs requires to evaluate the log-likelihood function \( L \) and gradient vector \( \{ \frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots \} \) at each training iteration. This is very time-consuming because estimating the partition function \( Z(o^{(j)}) \) and the expected value \( \mathbb{E}_{p_\theta} C_k(s, o^{(j)}) \) needs an intensive forward-backward computation. This computation manipulates on the transition matrix \( M_t \) at each time position \( t \) of each data sequence. \( M_t \) is defined as follows,

\[
M_t[t][l] = \exp \mathbf{F}(s, o, t) = \exp \left( \sum_j \lambda_j f_i(s_{t-1}, s_t) + \sum_j \lambda_j g_i(o, s_t) \right) \tag{7}
\]

To compute the partition function \( Z(o^{(j)}) \) and the expected value \( \mathbb{E}_{p_\theta} C_k(s, o^{(j)}) \), we need forward and backward vector variables \( \alpha_t \) and \( \beta_t \) defined as follows,

\[
\alpha_t = \begin{cases} \alpha_{t+1} M_t & 0 < t \leq T \\ 1 & t = 0 \end{cases} \tag{8}
\]

\[
\beta_t^T = \begin{cases} M_{t+1}^T \beta_{t+1}^T & 1 \leq t < T \\ 1 & t = T \end{cases} \tag{9}
\]

\[
Z(o^{(j)}) = \alpha_T \beta_T^T \tag{10}
\]

\[
\mathbb{E}_{p_\theta} C_k(s, o^{(j)}) = \sum_{t=1}^T \alpha_{t-1}(f_k * M_t) \beta_t^T \tag{11}
\]

### 4.1 High-Performance Parallel Training of Conditional Random Fields

#### 4.1.1 The Need of Parallel Training of CRFs

In the sequential algorithm for training CRFs in Table 1, step (1) is most time-consuming. This is because of the heavy forward-backward computation on transition matrices to estimate the log-likelihood function \( L \) and its gradient \( \{ \frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots \} \). The L-BFGS update, i.e., step (2), is very fast even if the log-likelihood function is very high dimensional, i.e., the CRF model contains up to millions of features. Therefore, the computational complexity of the training algorithm is mainly estimated from step (1).

The time complexity for calculating the transition matrix \( M_t \) in (7) is \( \mathcal{O}(n|L|^2) \) where \(|L|\) is the number of class labels and \( n \) is the average number of active features at a time position in a data sequence. Thus,
the time complexity to the partition function \( Z(\mathbf{o}^{(j)}) \) according to (8) and (10) is \( O(n|\mathcal{L}|^2T) \), in which \( T \) is the length of the observation sequence \( \mathbf{o}^{(j)} \). And, the time complexity for computing the feature expectation \( E_{\mathcal{P}_n} C_k(s, \mathbf{o}^{(j)}) \) is also \( O(n|\mathcal{L}|^2T) \). As a result, the time complexity for evaluating the log-likelihood function and its gradient vector is \( O(Nn|\mathcal{L}|^2T) \), in which \( N \) is the number of training data sequences and \( T \) is now replaced by \( T \) - the average length of training data sequences. Because we train the CRF model \( m \) iterations, the final computational complexity of the serial training algorithm is \( O(mNn|\mathcal{L}|^2T) \). This computational complexity is for first-order Markov CRFs. If we use the second-order Markov CRFs in which the label of the current state depends on two labels of two previous states, the complexity is now proportional to \( |\mathcal{L}|^4 \), i.e., \( O(mNn|\mathcal{L}|^4T) \).

Although the training complexity of CRFs is polynomial with respect to all input parameters, the training process on large-scale datasets is still prohibitively expensive. In practical implementation, the computational time for training CRF is even larger than what we can estimate from the theoretical complexity: this is because many other operations need to be performed during training, such as feature scanning, mapping between different data formats, numerical scaling (to avoid numerical problems), and smoothing. For example, training a first-order Markov CRF model for POS tagging (\(|\mathcal{L}| = 45\)) on about 1 million words (i.e., \( NT \approx 1,000,000 \)) of the Wall Street Journal corpus (Penn TreeBank) took approximately 100 hours, i.e., more than 4 days.

Further, the learning performance of a model depends on different parameter settings. As a result, the training procedure should be performed over and over till it reaches the desired optimum. Additionally, feature induction methods for CRFs [15, 16] usually take hours to induce the most useful features from millions of candidates. For those reasons, we decided to implement an efficient training procedure for CRFs on massively parallel processing systems that can reduce the training time dramatically. This enables us to evaluate CRFs with higher-order Markov dependencies on very large corpora - that cannot be done before.

4.2 The Parallel Training of CRFs

As we can see from (5) and (6), the log-likelihood function and its gradient vector with respect to training dataset \( \mathcal{D} \) are computed by summing over all training data sequences. This nature sum allows us to divide the training dataset into different partitions and evaluate the log-likelihood function and its gradient on each partition independently. As a result, the parallelization of the training process is quite straightforward.

4.2.1 How the Parallel Algorithm Works

The parallel algorithm is shown in Table 2. The algorithm follows the master-slave strategy. In this algorithm, the training dataset \( \mathcal{D} \) is randomly divided into \( P \) equal partitions: \( \mathcal{D}_1, \ldots, \mathcal{D}_P \). At the initialization step, each data partition is loaded into the internal memory of its corresponding process. Also, every process maintains the same vector of feature weights \( \theta \) in its internal memory.

At the beginning of each training iteration, the vector of feature weights on each process will be updated by communicating with the master process. Then, the local log-likelihood \( L_i \) and gradient vector \( \{ \frac{\delta L}{\lambda_1}, \frac{\delta L}{\lambda_2}, \ldots \} \) are evaluated in parallel on distributed processes; the master process then gathers and sums those values to obtain the global log-likelihood \( L \) and gradient vector \( \{ \frac{\delta L}{\lambda_1}, \frac{\delta L}{\lambda_2}, \ldots \} \); the new setting of feature weights is updated on the master process using L-BFGS optimization. The algorithm will check for some terminating criteria to whether stop or perform the next iteration. The output of the training process is the optimal vector of feature weights \( \theta^* = \{ \lambda_1^*, \lambda_2^*, \ldots \} \).

### Table 2 Parallel algorithm for training CRFs

<table>
<thead>
<tr>
<th>Input:</th>
<th>- Training data: ( \mathcal{D} = { (o^{(j)}, l^{(j)}) }_{j=1}^{N} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- The number of parallel processes: ( P )</td>
</tr>
<tr>
<td></td>
<td>- The number of training iterations: ( m )</td>
</tr>
<tr>
<td>Output:</td>
<td>- Optimal feature weights: ( \theta^* = { \lambda_1^<em>, \lambda_2^</em>, \ldots } )</td>
</tr>
<tr>
<td>Initial Step:</td>
<td>- Generate features with initial weights ( \theta = { \lambda_1, \lambda_2, \ldots } )</td>
</tr>
<tr>
<td></td>
<td>- Each process loads its own data partition ( \mathcal{D}_i )</td>
</tr>
<tr>
<td>Parallel Training (each training iteration):</td>
<td>1. The root process broadcasts ( \theta ) to all parallel processes</td>
</tr>
<tr>
<td></td>
<td>2. Each process ( P_i ) computes the local log-likelihood ( L_i ) and local gradient vector ( { \frac{\delta L}{\lambda_1}, \frac{\delta L}{\lambda_2}, \ldots } ) on ( \mathcal{D}_i )</td>
</tr>
<tr>
<td></td>
<td>3. The root process gathers and sums all ( L_i ) and ( { \frac{\delta L}{\lambda_1}, \frac{\delta L}{\lambda_2}, \ldots } ), to obtain the global ( L ) and ( { \frac{\delta L}{\lambda_1}, \frac{\delta L}{\lambda_2}, \ldots } )</td>
</tr>
<tr>
<td></td>
<td>4. The root process performs L-BFGS optimization search to update the new feature weights ( \theta )</td>
</tr>
<tr>
<td></td>
<td>5. If #iterations &lt; ( m ) then goto step 1, stop otherwise</td>
</tr>
</tbody>
</table>

4.2.2 Data Communication and Synchronization

In each training iteration, the master process has to communicate with each slave process twice: (1) broadcasting the vector of feature weights and (2) gathering the local log-likelihood and gradient vector. These operations are performed using message passing mechanism. Let \( n \) be the number of feature weights and weights are encoded with “double” data type, the total amount of data needs to be transferred between the master and each slave is \( 8(2n + 1) \). If, for example, \( n = 1,500,000 \), the amount of data is approximately 23Mb. This is very small in comparison with high-speed links among computing nodes on massively parallel processing systems. A barrier synchronization is needed at each training iteration to wait for all processes complete their estimation of local log-likelihood and gradient vector.
4.2.3 Data Partitioning and Load Balancing

Load balancing is important to parallel programs for performance reasons. Because all tasks are subject to a barrier synchronization point at each training iteration, the slowest process will determine the overall performance. In order to keep a good load balance among processes, i.e., to reduce the total idle time of computing processes as much as possible, we attempt to divide data into partitions as equally as possible. Let $M = \sum_{j=1}^{N} |o(j)|$ be the total number of data observations in training dataset $D$. Ideally, each data partition $D_i$ consists of $N_i$ data sequences having exactly $\frac{M}{N_i}$ data observations. However, this ideal partitioning is not always easy to find because the lengths of data sequences are different. To simplify the partitioning step, we accept an approximate solution as follows. Let $\delta$ be some integer number, we attempt to find a partitioning in which the number of data observations in each data partition belongs to the interval $[\frac{M}{N} - \delta, \frac{M}{N} + \delta]$. To search for the first acceptable solution, we follow the round-robin partitioning policy in which longer data sequences are considered first. $\delta$ starts from some small value and will be gradually increased until the first solution is satisfied.

5. Empirical Evaluation

We performed two important natural language processing tasks, text noun phrase chunking and all-phrase chunking, on large-scale datasets to demonstrate two main points: (1) the large reduction in computational time of the parallel training of CRFs on massively parallel computers in comparison with the serial training; (2) when being trained on large-scale datasets, CRFs tends to achieve higher prediction accuracy in comparison with the previous applied learning methods.

5.1 Experimental Environment

The experiments were carried out using our C/C++ implementation, PCRFs\(^\dagger\), of second-order Markov CRFs. It was designed to deal with hundreds of thousand data sequences and millions of features. It can be compiled and run on any parallel system supporting message passing interface (MPI). We used a Cray XT3 system (Linux OS, 180 AMD Opteron 2.4GHz processors, 8GB RAM per each, high-speed (7.6GB/s) interconnection passing interface (MPI). We used a Cray XT3 system and run on any parallel system supporting message passing to avoid load imbalances.

5.2 Text Chunking

Text chunking\(^\text{††}\), an intermediate step towards full parsing of natural language, recognizes phrase types (e.g., noun phrase, verb phrase, etc.) in input text sentences.

Here is a sample sentence with phrase marking: “[NP Rolls-Royce Motor Cars Inc.] [VP said] [NP [if] [NP [its] U.S. sales] [VP to remain] [ADJP steady] [VP [about 1,200 cars]] [PP in] [NP [1999].]” We evaluate two main tasks: noun phrase chunking (NP chunking for short) and all-phrase chunking (chunking for short) with different data sizes and parameter configurations.

5.3 Text Chunking Data and Evaluation Metric

We evaluated NP chunking and all-phrase chunking on the two data configurations as follows: (1) CoNLL2000-L: the training dataset consists of 39,832 sentences of sections from 02 to 21 of the Wall Street Journal (WSJ) corpus (of Penn Treebank\(^\text{†††}\)) and the testing set includes 1,921 sentences of section 00 of WSJ; and (2) 25-fold CV Test: 25-fold cross-validation test on all 25 sections of WSJ. For each fold, we took one section of WSJ as the testing set and all the others as training set. For example, the testing set of the 2nd fold includes 1,993 sentences from section 01 and the training set includes 47,215 sentences from all the other sections.

Label representation for phrases is either IOB2 or IOE2. B indicates the beginning of a phrase, I is the inside of a phrase, E marks the end of a phrase, and O is outside of all phrases. The label path in IOB2 of the sample sentence is “B-NP LNP LNP B-VP B-NP B-VP B-NP LNP LNP B-VP LVP B-ADJP B-PP B-NP LNP LNP B-PP B-PP B-PP B-NP LNP B-PP B-NP O”.

Evaluation metrics are precision ($\text{pre.} = \frac{p}{r}$), recall ($\text{rec.} = \frac{r}{t}$), and $F_{\beta=1} = 2 \times (\text{pre.} \times \text{rec.})/(\text{pre.} + \text{rec.})$; in which $a$ is the number of correctly recognized phrases (by model), $b$ is the number of recognized phrases (by model), and $c$ is the number of actual phrases (by humans). We trained our CRF models using different initial values of feature weights ($\theta$) to examine how the starting point influences the learning performance (note that the expression $\theta = .01$ means $\theta = (.01, .01, .01, .01)$).

5.4 Feature Selection for Text Chunking

To achieve high prediction accuracy on these tasks, we train CRF model using the second-order Markov dependencies. This means that the label of the current state depends on the labels of the two previous states. As a result, we have four feature types as follows rather than only two types in first-order Markov CRFs.

\[
\begin{align*}
 f_i(s_{t-1}, s_t) &\equiv [s_{t-1} = l'][s_t = l] \\
 g_i(o, s_t) &\equiv [x_t(o, l)] [s_t = l] \\
 f_k(s_{t-2}, s_{t-1}, s_t) &\equiv [s_{t-2} = l'][s_{t-1} = l'][s_t = l] \\
 g_h(o, s_{t-1}, s_t) &\equiv [x_h(o, l)] [s_{t-1} = l'][s_t = l] \\
\end{align*}
\]

where $f_i$ and $g_i$ are the same as in first-order Markov CRFs; and $f_k$ and $g_h$ are the edge and state features that are only be used in second-order CRFs.

\(^\dagger\)The source code and document of PCRFs are available at http://www.jaist.ac.jp/~hienuxuan/lexcrfs/lexcrfs.html

\(^\text{††}\)For more information about text chunking task, see the shared task: http://www.cnts.ua.ac.be/conll2000/chunking

\(^\text{†††}\)Penn Treebank: http://www.cis.upenn.edu/~treebank
5.5 Experimental Results of Text Chunking

Table 4 shows the results of NP chunking and chunking, respectively.

Table 5 shows a comparison between NP chunking and chunking on the whole WSJ.

Table 6 shows an accuracy comparison between NP chunking and chunking on the whole WSJ.
ours and the state-of-the-art chunking systems on the CoNLL2000-L dataset. Sang [17] performed majority voting among classifiers and got an $F_{\beta=1}$ of 94.90. Kudo and Matsumoto [18] also reported voting $F_{\beta=1}$ of 95.77 using SVMs. No previous work reported results of chunking on this dataset. Our CRFs used from 1.3 to 1.5 million features and achieved $F_{\beta=1}$ scores of 96.59 and 96.18. We also voted among CRFs and obtained the best scores of 96.74 and 96.33, respectively. Our model reduces error by 22.93% on NP chunking relative to the previous best system.

5.6 Computational Time Measure and Analysis

We also measured the computational time of the CRF models the Cray XT3 system. Table 7 reports the training time for three tasks using a single process and parallel processes. For example, training 130 iterations of NP chunking task on CoNLL2000-L dataset using a single process took 38h57’ while it took only 56’ on 45 parallel processes. Similarly, each fold of the 25-fold CV test of NP chunking took an average training time of 1h21’ on 45 processes while it took approximately 56h on one process. All-phrase chunking is much more time-consuming. This is because the number of class labels is $|C| = 23$ on CoNLL2000-L. For example, serial training on the CoNLL2000-L requires about 1348h for 200 iterations (i.e., about 56 days) whereas it took only 17h46’ on 90 parallel processes.

<table>
<thead>
<tr>
<th>Task (iterations)</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP chunking (140)</td>
<td>38h57’</td>
</tr>
<tr>
<td>CV test of WSJ (150)</td>
<td>56h</td>
</tr>
<tr>
<td>Chunking (200)</td>
<td>134h46’</td>
</tr>
</tbody>
</table>

Table 7 Training time of the second-order CRF models on single process and parallel processes

Figure 2 depicts the computational time and the speed-up ratio of the parallel training CRFs on the Cray XT3 system. The left graph shows the significant reduction of computational time as a function of the number of parallel processes. The middle graph shows the left graph with log10 scale. The right graph shows the speed-up ratio when we increase the number of parallel processes. We can see that the real speed-up ratio (the lower line) approaches the theoretical speed-up line (the upper line). We observed that the time for L-BFGS search and data communication as well as synchronization at each training iteration is much smaller than the time for estimating the local log-likelihood values and its gradient vectors. This can explain why parallel training CRFs is so efficient.

6. Summary

We have presented a high-performance training of CRFs on large-scale datasets using massively parallel computers. And the empirical evaluation on text chunking with different data sizes and parameter configurations shows that second-order Markov CRFs can achieved a significantly higher accuracy in comparison with the previous results, particularly when being provided enough computing power and training data. And, the parallel training algorithm for CRFs helps reduce computational time dramatically, allowing us to deal with large-scale problems not limited to natural language processing.

References

Fig. 2 The computational time of parallel training and the speed-up ratio of the first fold (using IOB2) of 25-fold CV test on WSJ


Xuan-Hieu Phan received his B.S. and M.S. degrees in Information Technology from College of Technology, Vietnam National University, Hanoi (VNU) in 2001 and 2003, respectively. Currently, he is a Ph.D candidate in Graduate School of Information Science, Japan Advanced Institute of Science and Technology (JAIST). His research interests have been mainly concerned with Statistical and Structured Machine Learning Methods for Natural Language Processing, Information Extraction, Web/Text Mining, Association Rule Mining, and High-Performance Computing for Data Mining.

Le-Minh Nguyen received the B.S. degree in Information Technology from Hanoi University of Science, and the M.S. degree in Information Technology from Vietnam National University, Hanoi in 1998 and 2001, respectively. He received the Ph.D in Information Science from Graduate School of Information Science, JAIST in 2004. Currently, he is a Post-doc fellow with the Graduate School of Information Science, JAIST. His research interests include Text Summarization, Natural Language Understanding, Machine Translation, and Information Retrieval.

Yasushi Inoguchi received his B.E. degree from Department of Mechanical Engineering, Tohoku University in 1991, and received M.S degree and Ph.D from JAIST (Japan Advanced Institute of Science and Technology) in 1994 and 1997, respectively. He is currently an Associate Professor of Center for Information Science at JAIST. He was a research fellow of the Japan Society for the Promotion of Science from 1994 to 1997. He is also a researcher of PRESTO program of Japan Science and Technology Agency since 2002. His research interest has been mainly concerned with parallel computer architecture, interconnection networks, GRID architecture, and high performance computing on parallel machines. Dr. Inoguchi is a member of IEEE and IPS of Japan.

Susumu Horiguchi received his M.E and D.E degrees from Tohoku University in 1978 and 1981, respectively. He is currently a full professor in the Graduate School of Information Science, Tohoku University. He was a visiting scientist at the IBM Thomas J. Watson Research Center from 1986 to 1987 and a visiting professor at The Center for Advanced Studies, the University of Southwestern Louisiana and at the Department of Computer Science, Texas A&M University in the summers of 1994 and 1997. He was also a professor in the Graduate School of Information Science, JAIST. He has been involved in organizing many international workshops, symposia and conferences sponsored by the IEEE, ACM, IASTED, IEICE and IPS. His research interests have been mainly concerned with Interconnection Networks, Parallel Computing Algorithms, Massively Parallel Processing, Parallel Computer Architectures, VLSI/WSI Architectures, and Multi-Media Integral Systems, and Data Mining. Prof. Horiguchi is a senior member of the IEEE Computer Society, and a member of the IPS and IASTED. He is currently serving as Editors for IEICE Transaction on Information and Systems and for Journal of Interconnection Networks.