A Named Entity Recognition Method based on Decomposition and Concatenation of Word Chunks

Tomoya Iwakura† Hiroya Takamura‡ Manabu Okumura‡
†Fujitsu Laboratories Ltd.
iwakura.tomoya@jp.fujitsu.com
‡Precision and Intelligence Laboratory, Tokyo Institute of Technology
{takamura,oku}@pi.titech.ac.jp

Abstract

We propose a Named Entity (NE) recognition method in which word chunks are repeatedly decomposed and concatenated. We can obtain features from word chunks, such as the first word of a word chunk and the last word of a word chunk, which cannot be obtained in word-sequence-based recognition methods. However, each word chunk may include a part of an NE or multiple NEs. To solve this problem, we use the following operators: SHIFT for separating the first word from a word chunk, POP for separating the last word from a word chunk, JOIN for concatenating two word chunks, and REDUCE for assigning an NE label to a word chunk. We evaluate our method on a Japanese NE recognition data set that includes about 200,000 annotations of 191 types of NEs from over 8,500 news articles. The experimental results show that the training and processing speeds of our method are faster than those of a linear-chain structured perceptron and a semi-Markov perceptron while high accuracy is maintained.

1 Introduction

Named Entity (NE) recognition is a process by which the names of particular classes and numeric expressions are recognized in text. NEs include person names, locations, organizations, dates, times, and so on. NE recognition is one of the basic technologies used in text processing, including Information Extraction (IE), Question Answering (QA), and Information Retrieval (IR).

Supervised learning algorithms have been applied successfully to create NE recognizers. In the early stages, algorithms for training classifiers, including Maximum Entropy Models (Uchimoto et al., 2000), AdaBoost (Carreras et al., 2002), and Support Vector Machines (Yamada, 2007) were widely used. Recently, learning algorithms for structured prediction, such as linear-chain Conditional Random Fields (CRFs) (Lafferty et al., 2001), linear-chain structured perceptron (Collins, 2002a), semi-Markov perceptron (Cohen and Sarawagi, 2004), and semi-Markov Conditional Random Fields (Sarawagi and Cohen, 2004), have been widely used because of their good performances in terms of accuracy.

However, the computational cost of using these algorithms for structured prediction can become problematic when we handle a large number of types of NE classes. The computational cost of learning first-order-Markov models with linear-chain CRFs or structured perceptron is $O(K^2N)$, where K is the number of types of classes and N is the length of the sentence. Semi-Markov-based algorithms, such as semi-Markov perceptron and semi-Markov CRFs, enumerate NE candidates represented by word chunks in advance for capturing features such as the first word of a chunk and the last word of a chunk. Therefore, the computational cost of a semi-Markov perceptron is $O(KLN)$, where L is the upper bound length of the entities.

The computational cost might not be a big problem, when we use these learning algorithms to recognize a small number of types of NEs, such as the seven types in MUC (Grishman and Sundheim, 1996), the eight types in IREX (Committee, 1999), and the four types in the CoNLL shared task (Tjong Kim Sang and De Meulder, 2003). However, the computational cost will be higher than ever, when we recognize a large types of classes like Sekine's extended NE hierarchy that includes about 200 types of NEs for covering several types of needs of IE, QA, and IR (Sekine et al., 2002).

This paper proposes a word-chunk-based NE recognition method for creating fast NE recog-

Proceedings of the 5th International Joint Conference on Natural Language Processing, pages 828–836,
Chiang Mai, Thailand, November 8 – 13, 2011.

nizers while high accuracy is maintained by capturing rich features extracted from word chunks. Our method recognizes NEs from word-chunk sequences identified by a base chunker. When we use a base chunker with a computational cost of $O(KN)$ or lower than it, we can maintain the computational cost of our method as $O(KN)$. This is because the length of the word-chunk sequences is less than or equal to the sentence length N. In addition, our method can use features extracted from word chunks that cannot be obtained in word-based NE recognitions.

However, each word chunk may include a part of an NE or multiple NEs. To solve this problem, we use the following operators: SHIFT for separating the first word from a word chunk, POP for separating the last word from a word chunk, JOIN for concatenating two word chunks, and REDUCE for assigning an NE class label to a word chunk. Therefore, we call our method SHIFT-POP-JOIN-REDUCE parser (SPJR for short).

We demonstrate experimentally that the training and processing speeds of SPJR-based NE recognizers can be considerably faster than those of a linear-chain-perceptron and a semi-Markov-perceptron, while high accuracy is maintained.

2 SHIFT-POP-JOIN-REDUCE Parser

This section describes our method that recognizes NEs from word chunk sequences. We assume word chunk sequences are given by a base chunker which is described in Section 3.4.

2.1 Operators for Word Chunks

To recognize NEs from word chunks, we use SHIFT and POP for decomposing word chunks, JOIN for concatenating two word chunks, and REDUCE for assigning one of the defined NE class labels. In the following, $C = \langle C_1, ..., C_{|C|} \rangle$ denotes a word chunk sequence. cbw_j is the first word of C_j, and cew_j is the last word of C_j.

- REDUCE: This operator assigns one of the NE labels to a word chunk.
- POP: This operator separates the last word from a word chunk, and the separated word is treated as a new word chunk. POP is only applied to a word chunk consisting of more than one word. When POP is applied to C_j, the last word cew_j is separated from C_j. Indices of the following word chunks C_k ($j+1 \leq k \leq |C|$) are incremented by 1, and the separated word cew_j becomes new C_{j+1}. There is one exceptional procedure for POP to use as much initial chunk information as possible. If POP is successively applied to the j-th chunk, the separated words are concatenated and regarded as the $(j+1)$-th chunk. For example, if C_j consists of words "w x y z" and POP is applied to both y and z, then the C_{j+1} will be considered to be "y z".
- SHIFT: This operator separates the first word from a word chunk, and the separated word is treated as a new word chunk. SHIFT is only applied to a word chunk consisting of more than one word. When SHIFT is applied to C_j, the first word cbw_j is separated from C_j. Indices of the word chunks C_k ($j \leq k \leq |C|$) are incremented by 1, and the separated word cbw_j becomes new C_j.
- JOIN: This operator concatenates two adjacent word chunks. When JOIN is applied to C_j and C_{j+1}, C_j and C_{j+1} are concatenated for creating new C_j. Indices of the word chunks C_k ($j+2 \leq k \leq |C|$) are decremented by 1. To avoid an endless loop by generating previously processed word chunks, we forbid JOIN from occurring immediately after POP or SHIFT.

2.2 Training an NE Recognizer

The input to our training procedure is the word chunks of a base chunker along with the NE labels on those correct word chunks. To train an NE recognizer, we first generate training samples, and then run a machine-learning algorithm over the training samples. Figure 1 shows a pseudo-code of the procedure for generating training samples from the i-th input. $\{T_1, ... T_M\}$ is a set of training data consisting of M sentences.

$T_i = \langle T_{i,1}, ..., T_{i,M_i} \rangle$ ($1 \leq i \leq M$) is the i-th training input. $T_{i,j}$ ($1 \leq j \leq M_i$) is the j-th chunk of T_i, and $l(T_{i,j})$ is the NE label of $T_{i,j}$. If $T_{i,j}$ is not an NE, $l(T_{i,j})$ is O. The procedure runs as follows:

- (S0) We generate initial word chunks C from the word-sequence consisting of T_i with the given base chunker. We start to check the following steps from (S1) to (S5) for generating samples. The following process continues until all word chunks in T_i are processed.

```
# T_i = {T_{i,1}, ... T_{i,M_i}}: an input
# T_{i,j} (1 ≤ j ≤ M_i): a word chunk
# l(T_{i,j}): T_{i,j}'s NE label
# W_i: words in T_i
# C = {C_1, ... C_{|C|}}: word chunks
# l(cbw_j): the NE label of the last word of C_j
# l(cew_j): the NE label of the first word of C_j
# gen(C_j, OP): generate a training sample
#       for OPERATOR (OP) applied to C_i
# (S0) to (S5) correspond to those of Sec. 2.2.
GenerateTrainingSample(T_i)
# (S0) Generate a word chunk-sequence
# with a base chunker.
C = Chunking(W_i); j = 1;
while j ≤ M_i do
  if C_j == T_{i,j} then # (S1)
    gen(C_j, REDUCE = l(T_{i,j})); j++;
  else if l(cew_j) == O then # (S2)
    gen(C_j, POP); C = POP(C, j);
  else if l(cbw_j) == O then # (S3)
    gen(C_j, SHIFT); C = SHIFT(C, j);
  else if ( a word or words included in C_j
    are not the constituents of T_j) then # (S4)
    gen(C_j, POP); C = POP(C, j);
  else # (S5)
    gen(C_j, JOIN); C = JOIN(C, j);
  end if
end while
```

Figure 1: A pseudo code of the generation of training samples.

- (S1) If the current chunk C_j is equivalent to the correct chunk $T_{i,j}$, we generate a training sample for REDUCE=$l(T_{i,j})$. This means REDUCE for annotating a word chunk with $l(T_{i,j})$. Then we move to the next word chunk. ($j++$)

- (S2) If the label of the last word of the current chunk cew_j is "O", a training sample for applying POP to C_j is generated. Then POP is applied to C_j.

- (S3) If the label of the first word of the current chunk cbw_j is "O", a training sample for applying SHIFT to C_j is generated. Then SHIFT is applied to C_j.

- (S4) If a word or words included in C_j are not the constituents of $T_{i,j}$, a training sample for applying POP to C_j is generated.

- (S5) If all the above steps are not executed, the correct NE exists across more than one chunk. Therefore, we generate a sample of current word chunk for JOIN.

After generating training samples for all the inputs, we train a model from the training samples with a machine-learning algorithm.

2.3 An Example of Training

Consider the following training data T_i.

- [Mr.]$_O$ [Ken Ono]$_{PER}$ [went]$_O$ [skiing]$_O$

We first identify base chunks, and the result is as follows:

- [Mr. Ken] [Ono went] [skiing]

We denote this base chunking result as C. Word chunks are indicated by bracketing, and a current chunk is underlined.

We first compare $T_{i,1}$ and C_1. C_1 is not equivalent to $T_{i,1}$, and the NE label of the first word of C_1, "Mr.", is "O". Therefore, we generate a training sample for applying SHIFT to C_1 by (S3), and apply SHIFT to C_1. The current C would be the following.

- [Mr.] [Ken] [Ono went] [skiing]

We compare C_1 and $T_{i,1}$ again, and C_1 is equivalent to $T_{i,1}$. Therefore, we generate a training sample for applying REDUCE=O to C_1 by (S1), and move to the next word chunk.

- [Mr.] [Ken] [Ono went] [skiing]

C_2 is not equivalent to $T_{i,2}$, and C_2 does not satisfy (S1) to (S4). We generate a training sample of JOIN for C_2 by (S5), and apply JOIN to C_2 and C_3.

- [Mr.] [Ken Ono went] [skiing]

C_2 is still not equivalent to $T_{i,2}$, and the NE label of the last word "went" is "O". Therefore, we generate a training sample for applying POP to C_2 by (S2), and apply POP to C_2.

- [Mr.] [Ken Ono] [went] [skiing]

C_2 is equivalent to $T_{i,2}$, and a training sample for REDUCE=PER is generated. The remaining C_3 and C_4 are also equivalent to $T_{i,3}$ and $T_{i,4}$, respectively. Thus, two training samples for REDUCE=O are generated.

2.4 NE Recognition

Figure 2 shows a pseudo-code of our NE recognition method. When recognizing NEs from a given word sequence, we first identify an initial word

```
# W: an input word-sequence
# C = {C_1, ... C_|C|}: word chunks
# L = {L_1, ... L_|C|}: NE class labels of word chunks
# Model : a trained model
NERecognition(W, Model)
# Generate a word chunk-sequence
# with a base chunker. j is for C.
C = Chunking(W); j = 1;
while j ≤ |C| do
  # Select an operation. If REDUCE is
  # selected, selectOP returns an NE label l.
  (op, l) = selectOP(C_j,Model);
  if op == REDUCE then
    L_j = l # keep C_j's NE label
    j++ # Move to the next word chunk
  else if op == POP then
    C = POP(C, j);
  else if op == SHIFT then
    C = SHIFT(C, j);
  else if op == JOIN then
    C = JOIN(C, j);
  end if
end while
return C and L
```

Figure 2: A pseudo code of NE recognition.

chunk sequence of the input with a base chunker as in the training.

Then we process each word chunk from the beginning of the sentence to the end of the sentence. An operator to use on the current word chunk is decided upon with a trained model, and each word chunk is processed according to the selected operator. If all the word chunks are processed, we return word chunks with their NE labels.

2.5 An Example of Recognition

Consider the following input data:

```
-Mr.  Jim Ji goes to U.K
```

We identify base chunks of the input as follows.

```
-[Mr.  Jim Ji] [goes] [to] [U.K]
```

We denote this base NE chunking result as C. An operator for each word chunk in C is selected with a trained model $Model$. Here, we assume SHIFT is selected and apply SHIFT to C_1. After applying SHIFT to C_1, C_1 becomes [Mr.], and C_2 becomes [Jim Ji].

```
-[Mr.] [Jim Ji] [goes] [to] [U.K]
```

We start to select an operator for the new C_1, and REDUCE=O is selected. We keep O as the NE class of C_1, and move to the next chunk C_2.

```
-[Mr.]  [Jim Ji] [goes] [to]
[U.K]
```

Next, we select an operator for the C_2, and REDUCE=PER is selected. We keep PER as the NE class of C_2, and move to the next chunk C_3. We continue this NE recognition process for the remaining word chunks. When we process all the word chunks, we return C with their NE labels.

3 Experimental Settings

3.1 Data Set and Evaluation Metrics

We used an extended NE corpus for our evaluation (Hashimoto et al., 2008). This Japanese corpus consists of about 8,500 articles from 2005 Mainichi newspaper. NE tags on this corpus is based on the extended NE hierarchy introduced by Sekine et al (Sekine et al., 2002). The corpus includes 240,337 tags for 191 types of NEs. To segment words from Japanese sentences, we used ChaSen.[1] We created the following sets for this experiment.

- training data: news articles from January to October 2005 in the corpus. The training data includes 1,806,772 words and 205,876 NEs.
- development data: news articles from November 2005 in the corpus. The development data includes 145,635 words and 15,405 NEs.
- test data: news articles from December 2005 in the corpus. The test data includes 177,159 words and 19,056 NEs.

Recall, precision, and F-measure are our evaluation metrics. Recall is defined to be the number of correctly recognized NEs divided by the number of all NEs. Precision is defined to be the number of correctly recognized NEs divided by the number of all recognized NEs. F-measure (FM) is defined as follows:

FM = 2 × recall × precision / (recall + precision).

[1] We use ChaSen-2.4.2 with Ipadic-2.7.0. ChaSen's web page is http://chasen-legacy.sourceforge.jp/. Words may include partial NEs because words segmented with ChaSen do not always correspond with NE boundaries. If such problems occur when we segment the training data, we annotated a word chunk with the type of the NE included in the word chunk. We did not deal with the difference between NE boundaries and word boundaries in this experiment.

3.2 Algorithms to be Compared

The following algorithms are compared with our method.

- Linear-chain structured perceptron (**Linear-Chain**, for short) (Collins, 2002a): This is a perceptron-based algorithm for labeling tags to word-sequences. In this algorithm, features are only generated from each word and its surrounding words.

- Semi-Markov perceptron (**Semi-Markov**, for short) (Cohen and Sarawagi, 2004): This algorithm is based on sequentially classifying chunks of several adjacent words, rather than single words. Ideally, all the possible word chunks of each input should be considered for this algorithm. However, the training of this algorithm requires a great deal of memory. Therefore, we limit the maximum length of the word-chunks. We use word chunks consisting of up to five or ten words. [2]

- NE Chunking and Classification (**NECC**, for short) (Carreras et al., 2002): This method consists of two parts. The first part is a base NE recognition as in our method. The second part is NE classification. Unlike in our method, this method just classifies given word chunks without decomposing and concatenating them. This method was used in the best system of the shared task of CoNLL 2002 (Tjong Kim Sang, 2002).

- Shift-Reduce Parser for NE Recognition (**SR**, for short) (Yamada, 2007): This algorithm is based on shift-reduce parsing for word-sequences. It uses two operators. The first one is shift which concatenates a word and its following word chunk. The other is reduce for annotating an NE label to current word chunk.[3] The algorithm is different from ours in that the initial inputs of their method are word sequences. Thus, each word chunk is constructed little by little. Therefore, the algorithm cannot use features obtained from word chunks at the early stage.

We use the multiclass perceptron algorithm for NECC, SR and SPJR. Thus all of the algorithms are based on perceptron (Rosenblatt, 1958). We apply the averaged perceptron (Collins, 2002a) for all the training algorithms. All the learners and NE recognizers were implemented with $C++$. We used perceptron-based algorithms because perceptron-based algorithms usually show the faster training speed and lower usage of memory than training algorithms, such as MEMM (McCallum et al., 2000), CRFs (Lafferty et al., 2001), and so on. Actually, when we applied a CRFs implementation based on LBFGS (Liu and Nocedal, 1989) to the training data, the implementation consumed 72GB memory which is our machine memory size.

We select the number of the iteration that shows the highest F-measure in the development data for each NE recognizer. We set the maximum iteration number at 50.

3.3 Features

The features used in our experiment are shown in Table 1. As features for Linear-Chain perceptron, we used the following. Here, k denotes the current word position. w_k is the k-th word, and p_k is the Part-Of-Speech (POS) tag of k-th word. We used the word and the POS of the k-th word and the words in 2-word windows before and after the k-th word with the current NE-tag t_k and the NE tag t_{k-1} of the previous word. Each NE tag is represented as IOB1 (Ramshaw and Marcus, 1995). This representation uses three tags I, O and B, to represent the inside, outside and beginning of a chunk. B is only used at the beginning of a chunk which immediately follows another chunk that NE class is the same. Each tag is expressed with NE classes, like I-CL, B-CL, where CL is an NE class.[4] To realize a fast training speed for Linear-Chain, we only used the valid combination of t_k and t_{k-1} in terms of the chunk representation.

[2] This is because when we ran Semi-Markov without the chunk length constraint, it used 72GB memory, which is our machine memory size, and 1 GB swap region on its hard disc.

[3] To compare the performance under the same conditions, we did not use the operator to separate characters from words to recognize partial NEs in words.

[4] We compared five types of chunk representation: IOB1 , IOB2, IOE1, IOE2 (Tjong Kim Sang and Veenstra, 1999) and Start/End (SE) (Uchimoto et al., 2000) in terms of the number of the NE tags. The number of the NE tags for each representation is as follows; IOB1 is 202, IOB2 is 377, IOE1 is 202, IOE2 is 377, and SE is 730. This experiment uses IOB1 because IOB1 has one of the lowest number of NE tags. The number of NE tags is related to the training speed of Linear-Chain. Actually, Linear-Chain using IOB1-based training data was about 2.4 times faster than Linear-Chain using SE-based training data in our pilot study with small training data.

Table 1: Features. k denotes a word positions for Linear-Chain. w_k is the k-th word surface, and p_k is the POS tag of the k-th word. t_k is the tag of the k-th word. TT_k is t_k or the combination of t_k and t_{k-1}. bp is the position of the first word of the current chunk. ep indicates the position of the last word of the current chunk. ip is the position of words inside the current chunk. ($bp < ip < ep$). If the length of the current chunk is 2, we use features that indicate there is no inside word as the features of ip-th words. t_j is the NE class label of j-th chunk. CL_j is the length of the current chunk, whether it be 1, 2, 3, 4, or longer than 4. WB_j indicates word bigrams, and PB_j indicates POS bigrams inside the current chunk.

Without chunk (Linear-Chain)
$[TT_k, w_k]$, $[TT_k, w_{k-1}]$, $[TT_k, w_{k-2}]$,
$[TT_k, w_{k+1}]$, $[TT_k, w_{k+2}]$, $[TT_k, p_k]$,
$[TT_k, p_{k-1}]$, $[TT_k, p_{k-2}]$, $[TT_k, p_{k+1}]$,
$[TT_k, p_{k+2}]$, $[TT_k, p_{k-2}, p_{k-1}]$,
$[TT_k, p_{k+1}, p_{k+2}]$, $[TT_k, p_{k-2}, p_{k-1}, p_{k+}]$,
$[TT_k, p_k, p_{k+1}, p_{k+2}]$
With chunk (Semi-Markov, NECC, SR, SPJR)
$[t_j, CL_j]$, $[t_j, WB_j]$, $[t_j, PB_j]$,
$[t_j, w_{bp}]$, $[t_j, p_{bp}]$,$[t_j, w_{ep}]$, $[t_j, p_{ep}]$,
$[t_j, w_{ip}]$,$[t_j, p_{ip}]$, $[t_j, w_{bp}, w_{ep}]$, $[t_j, p_{bp}, p_{ep}]$,
$[t_j, w_{bp}, p_{ep}]$, $[t_j, p_{bp}, w_{ep}]$,
$[t_j, w_{bp-1}]$, $[t_j, p_{bp-1}]$, $[t_j, w_{bp-2}]$, $[t_j, p_{bp-2}]$,
$[t_j, w_{ep+1}]$, $[t_j, p_{ep+1}]$, $[t_j, w_{ep+2}]$, $[t_j, p_{ep+2}]$,
$[t_j, p_{bp-2}, p_{bp-1}]$, $[t_j, p_{ep+1}, p_{ep+2}]$,
$[t_j, p_{bp-2}, p_{bp-1}, p_{bp}]$, $[t_j, p_{ep}, p_{ep+1}, p_{ep+2}]$

Semi-Markov, NECC, SR, and SPJR, using word chunks, used features extracted from words in a word chunk and the words in two-word windows before and after the word chunk. The features extracted from chunks differ from those of Linear-Chain.

3.4 Base Chunkers

NECC and SPJR require a base chunker. To compare performances obtained with different base chunkers, we used a rule-based chunker and a machine-learning-based chunker.

We used a chunker that concatenates successive words with noun or unknown POS tags, or words existing in brackets, for the rule based chunker (**RC**, for short). This identification is fast because the chunker only checks POS tags.

Our machine-learning-based base chunker is the SR-based chunker trained to distinguish just two chunks: NE and non-NE. To train a machine-learning-based chunker, we first converted the given training data into a training data for base chunkers. There are only two classes for the training data used for base chunking: NE or not.

For example, the following labeled input,
`Dr. [Toru Tanaka]`$_{PER}$ `goes to [Kyoto]`$_{LOC}$
is converted as follows.
`[Dr.]`$_O$ `[Toru Tanaka]`$_{BNE}$ `[goes]`$_O$ `[to]`$_O$ `[Kyoto]`$_{BNE}$
BNE indicates that the word chunk becomes NE, and O indicates non-NE.

The converted training data are used for training a base NE chunker that identifies base NEs and non-NEs. Words identified as non-NEs are treated as word chunks consisting of a word.

Then we split the converted training data into five portions. To obtain a training set for recognizing NEs, we repeated the following process for all the five portions. We trained a base NE chunker with four out of five the portions of the converted training data. The base NE chunker identified base NEs from the remaining portion. After all the portions were processed, we trained the SPJR- or NECC-based NE recognizer from the five portions along with the NE labels on those correct word chunks.

For recognizing NEs in the test phase, we used a base NE chunker trained with all the given training data converted by the method. To examine whether we can attain high accuracy while maintaining fast training and processing speed, we used SR, the fastest algorithm among our compared algorithms, for training base NE chunkers.

4 Experimental Results

Table 2 shows the experimental results.[5] The NE recognizers based on our method showed good performance. SPJR with RC is the second, and SPJR with SR is the fourth best F-measure on test data. The best and the third systems are Semi-Markov-based ones. These results indicate that features extracted from word chunks contributed to improved accuracy.

To compare the results of SPJR (RC) with the others, we employed a McNemar paired test on the labeling disagreements as was done in (Sha and

[5]We used a machine with Intel(R) Xeon(R) CPU X5680 @ 3.33GHz and 72 GB memory.

Table 2: Experimental results. FM, RE, and PR indicate F-measure Recall, and Precision, obtained with each algorithm, respectively. The dev. and test indicate the results for the development data, and the results for the test data. MEM. indicates the amount of memory size (GB) for the training of each algorithm. TRAIN. indicates the training time for each algorithm (in hours). PROC. indicates the processing speed of the development data for each algorithm (in seconds). (SR) and (RC) indicate NECC or SPJR that uses a SR-based chunker or RC, respectively. The numbers after L= indicate the maximum length of the word-chunks for Semi-Markov perceptron.

Algorithm	FM (RE,PR) for dev.	FM (RE,PR) for test	MEM.	TRAIN.	PROC.
Linear-Chain	78.95 (75.53, 82.68)	80.62 (77.36, 84.18)	12.9	23.54	81.27
Semi-Markov (L=5)	79.46 (76.78, 82.34)	80.90 (78.39, 83.58)	9.8	0.97	41.06
Semi-Markov (L=10)	80.54 (77.46, 83.86)	81.95 (79.04, 85.08)	18.3	1.98	62.96
SR	78.66 (74.53, 83.28)	79.89 (75.84, 84.39)	0.78	0.12	3.54
NECC (SR)	78.81 (72.02, 87.01)	80.61 (73.98, 88.55)	0.76	0.26	5.75
NECC (RC)	51.56 (36.73, 86.49)	50.00 (35.09, 86.93)	0.73	0.08	2.98
SPJR (SR)	79.30 (76.06, 82.82)	80.83 (77.85, 84.05)	0.76	0.29	5.86
SPJR (RC)	79.48 (76.05, 83.23)	81.09 (77.76, 84.71)	3.05	0.27	3.02

Pereira, 2003). We compared results on test data by character units because the ends or beginnings of Japanese NEs do not always correspond with word boundaries. All the results except for Semi-Markov (L=10) indicate that there is a significant difference ($p < 0.01$). This result shows that SPJR (RC) showed high accuracy.

Our method also showed faster training and processing speeds than those of Linear-Chain and Semi-Markov. Specifically, our proposed method with RC showed about a 87 times faster training speed and about a 27 times faster processing speed than those of Linear-Chain. Our method showed about a 7 times faster training speed and about a 21 times faster processing speed than those of Semi-Markov (L=10). In addition, our algorithm requires a lower maximum memory size than Linear-Chain and Semi-Markov.

5 Discussion

5.1 Comparison with Linear-Chain

Our algorithm showed much faster speed than Linear-Chain. This is because the difference of computational procedure. Linear-Chain-based ones run Viterbi algorithm to select the best labeled sequence in terms of the scores assigned by a trained model for each input in both training and testing. When running Viterbi algorithm, Linear-Chain-based ones have to check the connections of class labels. The number of connections is up to K^2, where K is the number of types of class labels. On the other hand, SPJR-based ones greedily recognize NEs.

5.2 Comparison with Semi-Markov

Since the maximum length of the word-chunks affects the performance of Semi-Markov, we evaluated Semi-Markov using two types of the maximum lengths of the word-chunks. Semi-Markov (L=5) showed faster training and processing speed than Semi-Markov (L=10). However, SPJR-based ones still showed much faster training and processing speed than Semi-Markov (L=5). In addition, compared with Semi-Markov (L=5), SPJR (RC) showed higher accuracy and SPJR (SR) showed competitive accuracy. This result indicates that SPJR-based ones show faster speed than Semi-Markov giving up accuracy for improving speed.

5.3 Comparison with SR

SR showed faster training and processing speeds than those of SPJR and NECC using the SR-based base NE chunker. Specifically, SR showed about a 2.4 times faster training speed and about a 1.6 times faster processing speed than those of SPJR. This is because SPJR and NECC require training for the SR-based NE chunker and base NE recognition. However, SPJR showed better accuracy than SR. In addition, SPJR with RC showed faster processing speed than SR. These results indicate that our method using an appropriate base chunker realizes fast processing speed while maintaining accuracy.

5.4 Comparison with NECC

SPJR with RC showed much higher accuracy than NECC with RC. This is because the accuracy of RC-based NEs recognition is low. The SR-based base NE chunker identifies 86.56% of word chunks that become NEs in development data. However, RC identified only 50.75%. NECC can only identify correct NEs from correct chunks. Therefore, NECC with RC showed low accuracy. These results indicate our method based on decomposition and concatenation of word chunks contributed to improving accuracy.

However, SPJR (RC) showed more memory usage than SPJR (SR) and NECC-based ones. This is also related to the accuracy of each base chunker. To correctly recognize NEs from word chunks wrongly identified by the rule-based chunker, more training samples for decomposition or concatenation were generated. NECC showed slightly faster training and processing speeds than SPJR. This may be because NECC identifies the NE class of each word chunk without decomposition or concatenation.

5.5 Computational Efficiency

Our algorithm showed faster training and processing speeds than Linear-Chain and Semi-Markov. Formally, the computational cost of Semi-Markov is $O(KLN)$, where L is the upper bound length of word chunks, N is the length of the sentence and K is the size of the label set. As for Semi-Markov, L is 10 and K is 191 in this experiment. And that of the first order Linear-Chain perceptron is $O(K^2N)$, and K is 202 in this experiment. In contrast, the computational cost of NECC, SR, and SPJR is $O(KN)$ and K is 191.

Semi-Markov required more memory usage than the other algorithms. The number of word chunks of each sentence in Semi-Markov is roughly LN. In contrast, the number of word chunks or words of each sentence for Linear-Chain, NECC, SR, and SPJR are up to N. This indicates that Semi-Markov handles about L times larger space than the other algorithms in terms of the number of nodes. Therefore, it requires more memory usage than the others.

6 Related Work

There have been methods proposed to improve the training speed for semi-Markov-based models and perceptron-based methods. With regard to reducing the space of lattices built into the Semi-Markov-based algorithms, a method was proposed to filter nodes in the lattices with a naive Bayes classifier (Okanohara et al., 2006). To improve training speed of the structured perceptron, distributed training strategies and convergence bounds for a particular mode of distributed the structured perceptron training are provided in (McDonald et al., 2010).

While the formulation is different, SPJR shares similar idea with transformation based learning (Brill, 1995). For example, a transformation-based POS tagging alters the POS tag of each word with an ordered list of transformations. These transformations alter the POS tag of each word based on contextual cues.

Methods using rich information other than semi-Markov-based algorithms are proposed as well. Previous works use N-best outputs to obtain rich features. For example, a boosting-based algorithm and a perceptron-based algorithm for reranking N-best outputs were proposed (Collins, 2002b). Another approach uses feature forests generated from N-best outputs (Huang, 2008). This method merges N-best outputs into a single lattice. In contrast with these methods, which require a computational cost for processing N-best outputs, our method only handles an output.

Our proposed method showed good performance in this experiment, however, there is a drawback due to our recognizing strategy. Since NE recognizers based on our proposed method recognizes NEs greedily, only a mistake may affect later recognition process. In the future, we consider methods to incorporate techniques used in shift-reduce parsing (Huang and Sagae, 2010), like beam search or dynamic programming, into our recognition method for solving the problem.

7 Conclusion

We proposed a method for recognizing NEs from word chunk sequences. Our method uses operators for decomposing and concatenating word chunks. Experimental results showed training and processing speeds of our method are faster than those of linear-chain structured perceptron and semi-Markov perceptron while high accuracy is maintained. In the future we would like to evaluate our method with other tasks, such as NP chunking, and Text Chunking. We would also like to evaluate our method with different base chunkers.

References

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565.

Xavier Carreras, Lluís Màrques, and Lluís Padró. 2002. Named entity extraction using adaboost. In *Proc. of CoNLL'02*, pages 167–170.

William W. Cohen and Sunita Sarawagi. 2004. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *Proc. of KDD'04*, pages 89–98.

Michael Collins. 2002a. Discriminative training methods for Hidden Markov Models: theory and experiments with perceptron algorithms. In *Proc. of EMNLP'02*, pages 1–8.

Michel Collins. 2002b. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *In Proc. of ACL'02*, pages 489–496.

IREX Committee. 1999. *Proc. of the IREX workshop*.

Ralph Grishman and Beth Sundheim. 1996. Message understanding conference- 6: A brief history. In *Proc. of COLING'06*, pages 466–471.

Taiichi Hashimoto, Takashi Inui, and Koji Murakami. 2008. Constructing extended named entity annotated corpora. *IPSJ SIG Notes*, 2008(113):113–120.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proc. of ACL'10*, pages 1077–1086.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL'08*, pages 586–594.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML'01*, pages 282–289.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528.

Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. 2000. Maximum entropy markov models for information extraction and segmentation. In *ICML*, pages 591–598.

Ryan McDonald, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *Proc. of NAACL HLT'10*, pages 456–464.

Daisuke Okanohara, Yusuke Miyao, Yoshimasa Tsuruoka, and Jun ichi Tsujii. 2006. Improving the scalability of semi-markov conditional random fields for named entity recognition. In *Proc. of ACL'06*.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Proc. of the Third Workshop on Very Large Corpora*, pages 82–94. Association for Computational Linguistics.

Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. 65(6):386–408.

Sunita Sarawagi and William W. Cohen. 2004. Semi-markov conditional random fields for information extraction. In *Proc. of NIPS'04*.

Satoshi Sekine, Kiyoshi Sudo, and Chikashi Nobata. 2002. Extended named entity hierarchy. In *Proc. of LREC'02*.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proc. of NAACL HLT'03*, pages 134–141.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proc. of CoNLL-2003*, pages 142–147.

Erik Tjong Kim Sang and Jorn Veenstra. 1999. Representing text chunks. In *Proc. of EACL '99*, pages 173–179.

Erik F. Tjong Kim Sang. 2002. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proc. of CoNLL'02*, pages 155–158.

Kiyotaka Uchimoto, Qing Ma, Masaki Murata, Hiromi Ozaku, Masao Utiyama, and Hitoshi Isahara. 2000. Named entity extraction based on a maximum entropy model and transformation rules. In *Proc. of ACL 2000*, pages 326–335.

Hiroyasu Yamada. 2007. Shift-reduce chunking for japanese named entity extraction. *IPSJ SIG Notes*, 2007(47):13–18.