Learning with Lookahead: Can History-Based Models Rival Globally Optimized Models?

Yoshimasa Tsuruoka[†*] **Yusuke Miyao**[‡*] **Jun'ichi Kazama**[*]
[†] Japan Advanced Institute of Science and Technology (JAIST), Japan
[‡] National Institute of Informatics (NII), Japan
[*] National Institute of Information and Communications Technology (NICT), Japan
tsuruoka@jaist.ac.jp yusuke@nii.ac.jp kazama@nict.go.jp

Abstract

This paper shows that the performance of history-based models can be significantly improved by performing lookahead in the state space when making each classification decision. Instead of simply using the best action output by the classifier, we determine the best action by looking into possible sequences of future actions and evaluating the final states realized by those action sequences. We present a perceptron-based parameter optimization method for this learning framework and show its convergence properties. The proposed framework is evaluated on part-of-speech tagging, chunking, named entity recognition and dependency parsing, using standard data sets and features. Experimental results demonstrate that history-based models with lookahead are as competitive as globally optimized models including conditional random fields (CRFs) and structured perceptrons.

1 Introduction

History-based models have been a popular approach in a variety of natural language processing (NLP) tasks including part-of-speech (POS) tagging, named entity recognition, and syntactic parsing (Ratnaparkhi, 1996; McCallum et al., 2000; Yamada and Matsumoto, 2003; Nivre et al., 2004). The idea is to decompose the complex structured prediction problem into a series of simple classification problems and use a machine learning-based classifier to make each decision using the information about the past decisions and partially completed structures as features.

Although history-based models have many practical merits, their accuracy is often surpassed by globally optimized models such as CRFs (Lafferty et al., 2001) and structured perceptrons (Collins, 2002), mainly due to the *label bias problem*. Today, vanilla history-based models such as maximum entropy Markov models (MEMMs) are probably not the first choice for those who are looking for a machine learning model that can deliver the state-of-the-art accuracy for their NLP task. Globally optimized models, by contrast, are gaining popularity in the community despite their relatively high computational cost.

In this paper, we argue that history-based models are not something that should be left behind in research history, by demonstrating that their accuracy can be significantly improved by incorporating a *lookahead* mechanism into their decision-making process. It should be emphasized that we use the word "lookahead" differently from some literature on syntactic parsing in which lookahead simply means looking at the succeeding words to choose the right parsing actions. In this paper, we use the word to refer to the process of choosing the best action by considering different sequences of future actions and evaluating the structures realized by those sequences. In other words, we introduce a lookahead mechanism that performs a search in the space of future actions.

We present a perceptron-based training algorithm that can work with the lookahead process, together with a proof of convergence. The algorithm enables us to tune the weight of the perceptron in such a way that we can correctly choose the right action for the

State	Operation	Stack	Queue
0			I saw a dog with eyebrows
1	shift	I	saw a dog with eyebrows
2	shift	I saw	a dog with eyebrows
3	reduce$_L$	saw(I)	a dog with eyebrows
...			
4		saw(I) dog(a)	with eyebrows
5	shift	saw(I) dog(a) with	eyebrows
6	shift	saw(I) dog(a) with eyebrows	
7	reduce$_R$	saw(I) dog(a) with(eyebrows)	
8	reduce$_R$	saw(I) dog(a, with(eyebrows))	
5’	reduce$_R$	saw(I, dog(a))	with eyebrows
6’	shift	saw(I, dog(a)) with	eyebrows
7’	shift	saw(I, dog(a)) with eyebrows	
8’	reduce $_R$	saw(I, dog(a)) with(eyebrows)	
9’	reduce $_R$	saw(I, dog(a), with(eyebrows))	

Figure 1: Shift-reduce dependency parsing

current state at each decision point, given the information obtained from a search.

To answer the question of whether the history-based models enhanced with lookahead can actually compete with globally optimized models, we evaluate the proposed framework with a range of standard NLP tasks, namely, POS tagging, text chunking (a.k.a. shallow parsing), named entity recognition, and dependency parsing.

This paper is organized as follows. Section 2 presents the idea of lookahead with a motivating example from dependency parsing. Section 3 describes our search algorithm for lookahead and a perceptron-based training algorithm. Experimental results on POS tagging, chunking, named entity recognition, and dependency parsing are presented in Section 4. We discuss relationships between our approach and some related work in Section 5. Section 6 offers concluding remarks with some potential research directions.

2 Motivation

This section describes an example of dependency parsing that motivates the introduction of lookahead in history-based models.

A well-known history-based approach to dependency parsing is *shift-reduce parsing*. This algorithm maintains two data structures, *stack* and *queue*: A stack stores intermediate parsing results, and a queue stores words to read. Two operations (actions), *shift* and *reduce*, on these data structures construct dependency relations one by one.

For example, assume that we are given the following sentence.

I saw a dog with eyebrows.

In the beginning, we have an empty stack, and a queue filled with a list of input words (State 0 in Figure 1). The *shift* operation moves the left-most element of the queue to the stack. In this example, State 1 is obtained by applying *shift* to State 0. After the two *shift* operations, we reach State 2, in which the stack has two elements. When we have two or more elements in the stack, we can apply the other operation, *reduce*, which merges the two stack elements by creating a dependency relation between them. When we apply *reduce$_L$*, which means to have the left element as a dependent of the right element, we reach State 3: The word “I” has disappeared from the stack and instead it is attached to its head word “saw”.[1] In this way, the shift-reduce parsing constructs a dependency tree by reading words from the queue and constructing dependency relations on the stack.

[1]In Figure 1, $H(D_1, D_2, \ldots)$ indicates that $D_1, D_2, \ldots$ are the dependents of the head H.

Let's say we have now arrived at State 4 after several operations. At this state, we cannot simply determine whether we should *shift* or *reduce*. In such cases, conventional methods rely on a multi-class classifier to determine the next operation. That is, a classifier is used to select the most plausible operation, by referring to the features about the current state, such as surface forms and POSs of words in the stack and the queue.

In the lookahead strategy, we make this decision by referring to future states. For example, if we apply *shift* to State 4, we will reach State 8 in the end, which indicates that "with" attaches to "dog". The other way, i.e., applying *reduce*$_R$ to State 4, eventually arrives at State 9', indicating "with" attaches to "saw". These future states indicate that we were implicitly resolving PP-attachment ambiguity at State 4. While conventional methods attempt to resolve such ambiguity using surrounding features at State 4, the lookahead approach resolves the same ambiguity by referring to the future states, for example, State 8 and 9'. Because future states can provide additional and valuable information for ambiguity resolution, improved accuracy is expected.

It should be noted that Figure 1 only shows one sequence of operations for each choice of operation at State 4. In general, however, the number of potential sequences grows exponentially with the lookahead depth, so the lookahead approach requires us to pay the price as the increase of computational cost. The primary goal of this paper is to demonstrate that the cost is actually worth it.

3 Learning with Lookahead

This section presents our framework for incorporating lookahead in history-based models. In this paper, we focus on *deterministic* history-based models although our method could be generalized to non-deterministic cases.

We use the word "state" to refer to a partially completed analysis as well as the collection of historical information available at each decision point in deterministic history-based analysis. State transitions are made by "actions" that are defined at each state. In the example of dependency parsing presented in Section 2, a state contains all the information about past operations, stacks, and queues as well as the observation (i.e. the words in the sentence). The possible actions are *shift*, *reduce*$_R$, and *reduce*$_L$. In the case of POS tagging, for example, a state is the words and the POS tags assigned to the words on the left side of the current target word (if the tagging is conducted in the left-to-right manner), and the possible actions are simply defined by the POS tags in the annotation tag set.

```
 1: Input
 2:   $d$: remaining depth of search
 3:   $S_0$: current state
 4: Output
 5:   $S$: state of highest score
 6:   $v$: highest score
 7:
 8: function SEARCH($d$, $S_0$)
 9:   if $d = 0$ then
10:     return $(S_0, \boldsymbol{w} \cdot \phi(S_0))$
11:   $(S, v) \leftarrow$ (null, $-\infty$)
12:   for each $a \in$ POSSIBLEACTIONS$(S_0)$
13:     $S_1 \leftarrow$ UPDATESTATE($S_0$, $a$)
14:     $(S', v') \leftarrow$ SEARCH$(d-1, S_1)$
15:     if $v' > v$ then
16:       $(S, v) \leftarrow (S', v')$
17:   return $(S, v)$
```

Figure 2: Search algorithm.

3.1 Search

With lookahead, we choose the best action at each decision point by considering possible sequences of future actions and the states realized by those sequences. In other words, we need to perform a *search* for each possible action.

Figure 2 describes our search algorithm in pseudo code. The algorithm performs a depth-first search to find the state of the highest score among the states in its search space, which is determined by the search depth d. This search process is implemented with a recursive function, which receives the remaining search depth and the current state as its input and returns the state of the highest score together with its score.

We assume a linear scoring model, i.e., the score of each state S can be computed by taking the dot product of the current weight vector $\boldsymbol{w}$ and $\phi(S)$, the feature vector representation of the state. The

```
1:  Input
2:    $C$: perceptron margin
3:    $D$: depth of lookahead search
4:    $S_0$: current state
5:    $a_c$: correct action
6:
7:  procedure UPDATEWEIGHT($C, D, S_0, a_c$)
8:    $(a^*, S^*, v) \leftarrow (\text{null}, \text{null}, -\infty)$
9:    for each $a \in$ POSSIBLEACTIONS($S_0$)
10:     $S_1 \leftarrow$ UPDATESTATE($S_0$, $a$)
11:     $(S', v') \leftarrow$SEARCH($D, S_1$)
12:     if $a = a_c$ then
13:       $v' \leftarrow v' - C$
14:       $S_c^* \leftarrow S'$
15:     if $v' > v$ then
16:       $(a^*, S^*, v) \leftarrow (a, S', v')$
17:   if $a^* \neq a_c$ then
18:     $\boldsymbol{w} \leftarrow \boldsymbol{w} + \phi(S_c^*) - \phi(S^*)$
```

Figure 3: Perceptron weight update

scores are computed at each leaf node of the search tree and backed up to the root.[2]

Clearly, the time complexity of deterministic tagging/parsing with this search algorithm is $O(nm^{D+1})$, where n is the number of actions needed to process the sentence, m is the (average) number of possible actions at each state, and D is the search depth. It should be noted that the time complexity of k-th order CRFs is $O(nm^{k+1})$, so a history-based model with k-depth lookahead is comparable to k-th order CRFs in terms of training/testing time.

Unlike CRFs, our framework does not require the locality of features since it is history-based, i.e., the decisions can be conditioned on arbitrary features. One interpretation of our learning framework is that it trades off the global optimality of the learned parameters against the flexibility of features.

3.2 Training a margin perceptron

We adapt a learning algorithm for margin perceptrons (Krauth and Mezard, 1987) to our purpose of optimizing the weight parameters for the lookahead search. Like other large margin approaches such as support vector machines, margin perceptrons are known to produce accurate models compared to perceptrons without a margin (Li et al., 2002).

Figure 3 shows our learning algorithm in pseudo code. The algorithm is very similar to the standard training algorithm for margin perceptrons, i.e., we update the weight parameters with the difference of two feature vectors (one corresponding to the correct action, and the other the action of the highest score) when the perceptron makes a mistake. The feature vector for the second best action is also used when the margin is not large enough. Notice that the feature vector for the second best action is automatically selected by using a simple trick of subtracting the margin parameter from the score for the correct action (Line 13 in Figure 3).

The only difference between our algorithm and the standard algorithm for margin perceptrons is that we use the states and their scores obtained from lookahead searches (Line 11 in Figure 3), which are backed up from the leaves of the search trees. In Appendix A, we provide a proof of the convergence of our training algorithm and show that the margin will approach at least half the true margin (assuming that the training data are linearly separable).

As in many studies using perceptrons, we average the weight vector over the whole training iterations at the end of the training (Collins, 2002).

4 Experiments

This section presents four sets of experimental results to show how the lookahead process improves the accuracy of history-based models in common NLP tasks.

4.1 Sequence prediction tasks

First, we evaluate our framework with three sequence prediction tasks: POS tagging, chunking, and named entity recognition. We compare our method with the CRF model, which is one of the de facto standard machine learning models for such sequence prediction tasks. We trained L1-regularized first-order CRF models using the efficient stochastic gradient descent (SGD)-based training method presented in Tsuruoka et al. (2009). Since our main in-

[2]In actual implementation, it is not efficient to compute the score of a state from scratch at each leaf node. For most of the standard features used in tagging and parsing, it is usually straight-forward to compute the scores incrementally every time the state is updated with an action.

terest is not in achieving the state-of-the-art results for those tasks, we did not conduct feature engineering to come up with elaborate features—we simply adopted the feature sets described in their paper (with an exception being tag trigram features tested in the POS tagging experiments). The experiments for these sequence prediction tasks were carried out using one core of a 3.33GHz Intel Xeon W5590 processor.

The first set of experiments is about POS tagging. The training and test data were created from the Wall Street Journal corpus of the Penn Treebank (Marcus et al., 1994). Sections 0-18 were used as the training data. Sections 19-21 were used for tuning the meta parameters for learning (the number of iterations and the margin C). Sections 22-24 were used for the final accuracy reports.

The experimental results are shown in Table 1. Note that the models in the top four rows use exactly the same feature set. It is clearly seen that the lookahead improves tagging accuracy, and our history-based models with lookahead is as accurate as the CRF model. We also created another set of models by simply adding tag trigram features, which cannot be employed by first-order CRF models. These features have slightly improved the tagging accuracy, and the final accuracy achieved by a search depth of 3 was comparable to some of the best results achieved by pure supervised learning in this task (Shen et al., 2007; Lavergne et al., 2010).

The second set of experiments is about chunking. We used the data set for the CoNLL 2000 shared task, which contains 8,936 sentences where each token is annotated with the "IOB" tags representing text chunks. The experimental results are shown in Table 2. Again, our history-based models with lookahead were slightly more accurate than the CRF model using exactly the same set of features. The accuracy achieved by the lookahead model with a search depth of 2 was comparable to the accuracy achieved by a computationally heavy combination of max-margin classifiers (Kudo and Matsumoto, 2001). We also tested the effectiveness of additional features of tag trigrams using the development data, but there was no improvement in the accuracy.

The third set of experiments is about named entity recognition. We used the data provided for the BioNLP/NLPBA 2004 shared task (Kim et al., 2004), which contains 18,546 sentences where each token is annotated with the "IOB" tags representing biomedical named entities. We performed the tagging in the right-to-left fashion because it is known that backward tagging is more accurate than forward tagging on this data set (Yoshida and Tsujii, 2007).

Table 3 shows the experimental results, together with some previous performance reports achieved by pure machine leaning methods (i.e. without rule-based post processing or external resources such as gazetteers). Our history-based model with no lookahead was considerably worse than the CRF model using the same set of features, but it was significantly improved by the introduction of lookahead and resulted in accuracy figures better than that of the CRF model.

4.2 Dependency parsing

We also evaluate our method in dependency parsing. We follow the most standard experimental setting for English dependency parsing: The Wall Street Journal portion of Penn Treebank is converted to dependency trees by using the head rules of Yamada and Matsumoto (2003).[3] The data is split into training (section 02-21), development (section 22), and test (section 23) sets. The parsing accuracy was evaluated with auto-POS data, i.e., we used our lookahead POS tagger (depth = 2) presented in the previous subsection to assign the POS tags for the development and test data. Unlabeled attachment scores for all words excluding punctuations are reported. The development set is used for tuning the meta parameters, while the test set is used for evaluating the final accuracy.

The parsing algorithm is the "arc-standard" method (Nivre, 2004), which is briefly described in Section 2. With this algorithm, state S corresponds to a parser configuration, i.e., the stack and the queue, and action a corresponds to shift, reduce$_L$, and reduce$_R$. In this experiment, we use the same set of feature templates as Huang and Sagae (2010).

Table 4 shows training time, test time, and parsing accuracy. In this table, "No lookahead (depth = 0)" corresponds to a conventional shift-reduce parsing method without any lookahead search. The results

[3]Penn2Malt is applied for this conversion, while dependency labels are removed.

	Training Time (sec)	Test Time (sec)	Accuracy
CRF (L1 regularization & SGD training)	847	3	97.11 %
No lookahead (depth = 0)	85	5	97.00 %
Lookahead (depth = 1)	294	9	97.19 %
Lookahead (depth = 2)	8,688	173	97.19 %
No lookahead (depth = 0) + tag trigram features	88	5	97.11 %
Lookahead (depth = 1) + tag trigram features	313	10	97.22 %
Lookahead (depth = 2) + tag trigram features	10,034	209	97.28 %
Structured perceptron (Collins, 2002)	n/a	n/a	97.11 %
Guided learning (Shen et al., 2007)	n/a	n/a	97.33 %
CRF with 4 billion features (Lavergne et al., 2010)	n/a	n/a	97.22 %

Table 1: Performance of English POS tagging (training times and accuracy scores on test data)

	Training time (sec)	Test time (sec)	F-measure
CRF (L1 regularization & SGD training)	74	1	93.66
No lookahead (depth = 0)	22	1	93.53
Lookahead (depth = 1)	73	1	93.77
Lookahead (depth = 2)	1,113	9	93.81
Voting of 8 SVMs (Kudo and Matsumoto, 2001)	n/a	n/a	93.91

Table 2: Performance of text chunking (training times and accuracy scores on test data).

clearly demonstrate that the lookahead search boosts parsing accuracy. As expected, training and test speed decreases, almost by a factor of three, which is the branching factor of the dependency parser.

The table also lists accuracy figures reported in the literature on shift-reduce dependency parsing. Most of the latest studies on shift-reduce dependency parsing employ dynamic programing or beam search, which implies that deterministic methods were not as competitive as those methods. It should also be noted that all of the listed studies learn structured perceptrons (Collins and Roark, 2004), while our parser learns locally optimized perceptrons. In this table, our parser without lookahead search (i.e. depth = 0) resulted in significantly lower accuracy than the previous studies. In fact, it is worse than the deterministic parser of Huang et al. (2009), which uses (almost) the same set of features. This is presumably due to the difference between locally optimized perceptrons and globally optimized structured perceptrons. However, our parser with lookahead search is significantly better than their deterministic parser, and its accuracy is close to the levels of the parsers with beam search.

5 Discussion

The reason why we introduced a lookahead mechanism into history-based models is that we wanted the model to be able to avoid making such mistakes that can be detected only in later stages. Probabilistic history-based models such as MEMMs should be able to avoid (at least some of) such mistakes by performing a Viterbi search to find the highest probability path of the actions. However, as pointed out by Lafferty et al. (2001), the per-state normalization of probabilities makes it difficult to give enough penalty to such incorrect sequences of actions, and that is primarily why MEMMs are outperformed by CRFs.

Perhaps the most relevant to our work in terms of learning is the general framework for search and learning problems in history-based models proposed by Daumé III and Marcu (2005). This framework, called LaSO (Learning as Search Optimization), can include many variations of search strategies such as beam search and A* search as a special case. Indeed, our lookahead framework could be regarded as a special case in which each search node con-

	Training time (sec)	Test time (sec)	F-measure
CRF (L1 regularization & SGD training)	235	4	71.63
No lookahead (depth = 0)	66	4	70.17
Lookahead (depth = 1)	91	4	72.28
Lookahead (depth = 2)	302	7	72.00
Lookahead (depth = 3)	2,419	33	72.21
Semi-Markov CRF (Okanohara et al., 2006)	n/a	n/a	71.48
Reranking (Yoshida and Tsujii, 2007)	n/a	n/a	72.65

Table 3: Performance of biomedical named entity recognition (training times and accuracy scores on test data).

	Training time (sec)	Test time (sec)	Accuracy
No lookahead (depth = 0)	1,937	4	89.73
Lookahead (depth = 1)	4,907	13	91.00
Lookahead (depth = 2)	12,800	31	91.10
Lookahead (depth = 3)	31,684	79	91.24
Beam search ($k = 64$) (Zhang and Clark, 2008)	n/a	n/a	91.4
Deterministic (Huang et al., 2009)	n/a	n/a	90.2
Beam search ($k = 16$) (Huang et al., 2009)	n/a	n/a	91.3
Dynamic programming (Huang and Sagae, 2010)	n/a	n/a	92.1

Table 4: Performance of English dependency parsing (training times and accuracy scores on test data).

sists of the next and lookahead actions[4], although the weight updating procedure differs in several minor points. Daumé III and Marcu (2005) did not try a lookahead search strategy, and to the best of our knowledge, this paper is the first that demonstrates that lookahead actually works well for various NLP tasks.

Performing lookahead is a very common technique for a variety of decision-making problems in the field of artificial intelligence. In computer chess, for example, programs usually need to perform a very deep search in the game tree to find a good move. Our decision-making problem is similar to that of computer Chess in many ways, although chess programs perform min-max searches rather than the "max" searches performed in our algorithm. Automatic learning of evaluation functions for chess programs can be seen as the training of a machine learning model. In particular, our learning algorithm is similar to the supervised approach (Tesauro, 2001; Hoki, 2006) in that the parameters are optimized based on the differences of the feature vectors realized by the correct and incorrect actions.

In history-based models, the order of actions is often very important. For example, backward tagging is considerably more accurate than forward tagging in biomedical named entity recognition. Our lookahead method is orthogonal to more elaborate techniques for determining the order of actions such as easy-first tagging/parsing strategies (Tsuruoka and Tsujii, 2005; Elhadad, 2010). We expect that incorporating such elaborate techniques in our framework will lead to improved accuracy, but we leave it for future work.

6 Conclusion

We have presented a simple and general framework for incorporating a lookahead process in history-based models and a perceptron-based training algorithm for the framework. We have conducted experiments using standard data sets for POS tagging, chunking, named entity recognition and dependency parsing, and obtained very promising results—the accuracy achieved by the history-based models en-

[4]In addition, the size of the search queue is always truncated to one for the deterministic decisions presented in this paper. Note, however, that our lookahead framework can also be combined with other search strategies such as beam search. In that case, the search queue is not necessarily truncated.

hanced with lookahead was as competitive as globally optimized models including CRFs.

In most of the experimental results, steady improvement in accuracy has been observed as the depth of the search is increased. Although it is not very practical to perform deeper searches with our current implementation—we naively explored all possible sequences of actions, future work should encompass extending the depths of search space by introducing elaborate pruning/search extension techniques.

In this work, we did not conduct extensive feature engineering for improving the accuracy of individual tasks because our primary goal with this paper is to present the learning framework itself. However, one of the major merits of using history-based models is that we are allowed to define arbitrary features on the partially completed structure. Another interesting direction of future work is to see how much we could improve the accuracy by performing extensive feature engineering in this particular learning framework.

Appendix A: Convergence of the Learning Procedure

Let $\{x^i, a_c^i\}_{i=1}^K$ be the training examples where a_c^i is the correct first action for decision point x^i, and let $\mathcal{S}^i$ be the set of all the states at the leaves of the search trees for x_i generated by the lookahead searches and $\mathcal{S}_c^i$ be the set of all the states at the leaves of the search tree for the correct action a_c^i. We also define $\overline{\mathcal{S}^i} = \mathcal{S}^i \setminus \mathcal{S}_c^i$. We write the weight vector before the k-th update as $\boldsymbol{w}^k$. We define $S_c^* = \underset{S \in \mathcal{S}_c^i}{\operatorname{argmax}} \, \boldsymbol{w} \cdot \phi(S)$ and $\overline{S^*} = \underset{\overline{S} \in \overline{\mathcal{S}^i}}{\operatorname{argmax}} \, \boldsymbol{w} \cdot \phi(\overline{S})$[5]. Then the update rule can be interpreted as $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + (\phi(S_c^*) - \phi(\overline{S^*}))$. Note that this update is performed only when $\phi(S_c) \cdot \boldsymbol{w}^k - C < \phi(\overline{S^*}) \cdot \boldsymbol{w}^k$ for all $S_c \in \mathcal{S}_c$ since otherwise S^* in the learning algorithm cannot be a state with an incorrect first action. In other words, $\phi(S_c) \cdot \boldsymbol{w} - \phi(\overline{S^*}) \cdot \boldsymbol{w} \geq C$ for all $S_c \in \mathcal{S}_c$ after convergence.

Given these definitions, we prove the convergence for the separable case. That is, we assume the existence of a weight vector $\boldsymbol{u}$ (with $||\boldsymbol{u}|| = 1$), δ (> 0), and R (> 0) that satisfy:

$$\forall i, \forall S_c \in \mathcal{S}_c^i, \forall \overline{S} \in \overline{\mathcal{S}^i} \quad \phi(S_c) \cdot \boldsymbol{u} - \phi(\overline{S}) \cdot \boldsymbol{u} \geq \delta,$$
$$\forall i, \forall S_c \in \mathcal{S}_c^i, \forall \overline{S} \in \overline{\mathcal{S}^i} \quad ||\phi(S_c) - \phi(\overline{S})|| \leq R.$$

The proof is basically an adaptation of the proofs in Collins (2002) and Li et al. (2002). First, we obtain the following relation:

$$\begin{aligned} \boldsymbol{w}^{k+1} \cdot \boldsymbol{u} &= \boldsymbol{w}^k \cdot \boldsymbol{u} + (\phi(S_c^*) \cdot \boldsymbol{u} - \phi(\overline{S^*}) \cdot \boldsymbol{u}) \\ &= \boldsymbol{w}^k \cdot \boldsymbol{u} + \delta \geq \boldsymbol{w}^1 \cdot \boldsymbol{u} + k\delta = k\delta. \end{aligned}$$

Therefore, $||\boldsymbol{w}^{k+1} \cdot \boldsymbol{u}||^2 = ||\boldsymbol{w}^{k+1}||^2 \leq (k\delta)^2$ — (1). We assumed $\boldsymbol{w}^1 = \mathbf{0}$ but this is not an essential assumption.

Next, we also obtain:

$$\begin{aligned} ||\boldsymbol{w}^{k+1}||^2 &\leq ||\boldsymbol{w}^k||^2 + 2(\phi(S_c^*) - \phi(\overline{S^*})) \cdot \boldsymbol{w}^k \\ &\quad + ||\phi(S_c^*) - \phi(\overline{S^*})||^2 \\ &\leq ||\boldsymbol{w}^k||^2 + 2C + R^2 \\ &\leq ||\boldsymbol{w}^1||^2 + k(R^2 + 2C) = k(R^2 + 2C) \text{— (2)} \end{aligned}$$

Combining (1) and (2), we obtain $k \leq (R^2 + 2C)/\delta^2$. That is, the number of updates is bounded from above, meaning that the learning procedure converges after a finite number of updates. Substituting this into (2) gives $||\boldsymbol{w}^{k+1}|| \leq (R^2 + 2C)/\delta$ — (3).

Finally, we analyze the margin achieved by the learning procedure after convergence. The margin, $\gamma(\boldsymbol{w})$, is defined as follows in this case.

$$\begin{aligned} \gamma(\boldsymbol{w}) &= \min_{x_i} \min_{S_c \in \mathcal{S}_c^i, \overline{S} \in \overline{\mathcal{S}^i}} \frac{\phi(S_c) \cdot \boldsymbol{w} - \phi(\overline{S}) \cdot \boldsymbol{w}}{||\boldsymbol{w}||} \\ &= \min_{x_i} \min_{S_c \in \mathcal{S}_c^i} \frac{\phi(S_c) \cdot \boldsymbol{w} - \phi(\overline{S^*}) \cdot \boldsymbol{w}}{||\boldsymbol{w}||} \end{aligned}$$

After convergence (i.e., $\boldsymbol{w} = \boldsymbol{w}^{k+1}$), $\phi(S_c) \cdot \boldsymbol{w} - \phi(\overline{S^*}) \cdot \boldsymbol{w} \geq C$ for all $S_c \in \mathcal{S}_c$ as we noted. Together with (3), we obtain the following bound:

$$\begin{aligned} \gamma(\boldsymbol{w}) &\geq \min_{x_i} \frac{\delta C}{2C + R^2} \\ &= \frac{\delta C}{2C + R^2} = \left(\frac{\delta}{2}\right)\left(1 - \frac{R^2}{2C + R^2}\right) \end{aligned}$$

As can be seen, the margin approaches at least half the true margin, $\delta/2$ as $C \to \infty$ (at the cost of infinite number of updates).

[5] S_c^* and $\overline{S^*}$ depend on the weight vector at each point, but we omit it from the notation for brevity.

References

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, pages 111–118.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of ICML*, pages 169–176.

Yoav Goldbergand Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of NAACL-HLT*, pages 742–750.

Kunihito Hoki. 2006. Optimal control of minimax search results to learn positional evaluation. In *Proceedings of the 11th Game Programming Workshop (GPW)*, pages 78–83 (in Japanese).

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

Liang Huang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of EMNLP*, pages 1222–1231.

J.-D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, and N. Collier. 2004. Introduction to the bio-entity recognition task at JNLPBA. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 70–75.

W Krauth and M Mezard. 1987. Learning algorithms with optimal stability in neural networks. *Journal of Phisics A*, 20(11):L745–L752.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL*.

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

Thomas Lavergne, Olivier Cappé, and François Yvon. 2010. Practical very large scale CRFs. In *Proceedings of ACL*, pages 504–513.

Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz S. Kandola. 2002. The perceptron algorithm with uneven margins. In *Proceedings of ICML*, pages 379–386.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

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

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of CoNLL*, pages 49–56.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *ACL 2004 Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57.

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 *Proceedings of COLING/ACL*, pages 465–472.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP 1996*, pages 133–142.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of ACL*, pages 760–767.

Gerald Tesauro, 2001. *Comparison training of chess evaluation functions*, pages 117–130. Nova Science Publishers, Inc.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of HLT/EMNLP 2005*, pages 467–474.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of ACL-IJCNLP*, pages 477–485.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, pages 195–206.

Kazuhiro Yoshida and Jun'ichi Tsujii. 2007. Reranking for biomedical named-entity recognition. In *Proceedings of ACL Workshop on BioNLP*, pages 209–216.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graphbased and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, pages 562–571.