Guided Learning for Bidirectional Sequence Classification

Libin Shen
BBN Technologies
Cambridge, MA 02138, USA
lshen@bbn.com

Giorgio Satta
Dept. of Inf. Eng'g.
University of Padua
I-35131 Padova, Italy
satta@dei.unipd.it

Aravind K. Joshi
Department of CIS
University of Pennsylvania
Philadelphia, PA 19104, USA
joshi@seas.upenn.edu

Abstract

In this paper, we propose guided learning, a new learning framework for bidirectional sequence classification. The tasks of learning the order of inference and training the local classifier are dynamically incorporated into a single Perceptron like learning algorithm. We apply this novel learning algorithm to POS tagging. It obtains an error rate of 2.67% on the standard PTB test set, which represents 3.3% relative error reduction over the previous best result on the same data set, while using fewer features.

1 Introduction

Many NLP tasks can be modeled as a sequence classification problem, such as POS tagging, chunking, and incremental parsing. A traditional method to solve this problem is to decompose the whole task into a set of individual tasks for each token in the input sequence, and solve these small tasks in a fixed order, usually from left to right. In this way, the output of the previous small tasks can be used as the input of the later tasks. HMM and MaxEnt Markov Model are examples of this method.

Lafferty et al. (2001) showed that this approach suffered from the so called *label bias problem* (Bottou, 1991). They proposed Conditional Random Fields (CRF) as a general solution for sequence classification. CRF models a sequence as an undirected graph, which means that all the individual tasks are solved simultaneously. Taskar et al. (2003) improved the CRF method by employing the large margin method to separate the gold standard sequence labeling from incorrect labellings. However, the complexity of quadratic programming for the large margin approach prevented it from being used in large scale NLP tasks.

Collins (2002) proposed a Perceptron like learning algorithm to solve sequence classification in the traditional left-to-right order. This solution does not suffer from the label bias problem. Compared to the undirected methods, the Perceptron like algorithm is faster in training. In this paper, we will improve upon Collins' algorithm by introducing a bidirectional searching strategy, so as to effectively utilize more context information at little extra cost.

When a bidirectional strategy is used, the main problem is how to select the order of inference. Tsuruoka and Tsujii (2005) proposed the *easiest-first* approach which greatly reduced the computation complexity of inference while maintaining the accuracy on labeling. However, the easiest-first approach only serves as a heuristic rule. The order of inference is not incorporated into the training of the MaxEnt classifier for individual labeling.

Here, we will propose a novel learning framework, namely *guided learning*, to integrate classification of individual tokens and inference order selection into a single learning task. We proposed a Perceptron like learning algorithm (Collins and Roark, 2004; Daumé III and Marcu, 2005) for guided learning. We apply this algorithm to POS tagging, a classic sequence learning problem. Our system reports an error rate of 2.67% on the standard PTB test set, a relative 3.3% error reduction of the previous best system (Toutanova et al., 2003) by using fewer features. By using deterministic search, it obtains an error rate of 2.73%, a 5.9% relative error reduction

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 760–767,
Prague, Czech Republic, June 2007.

over the previous best deterministic algorithm (Tsuruoka and Tsujii, 2005).

The new POS tagger is similar to (Toutanova et al., 2003; Tsuruoka and Tsujii, 2005) in the way that we employ context features. We use a bidirectional search strategy (Woods, 1976; Satta and Stock, 1994), and our algorithm is based on Perceptron learning (Collins, 2002). A unique contribution of our work is on the integration of individual classification and inference order selection, which are learned simultaneously.

2 Guided Learning for Bidirectional Labeling

We first present an example of POS tagging to show the idea of bidirectional labeling. Then we present the inference algorithm and the learning algorithm.

2.1 An Example of POS tagging

Suppose that we have an input sentence

```
Agatha found that book interesting
  w1     w2    w3   w4      w5
             (Step 0)
```

If we scan from left to right, we may find it difficult to resolve the ambiguity of the label for `that`, which could be either DT (determiner), or IN (preposition or subordinating conjunction) in the Penn Treebank. However, if we resolve the labels for `book` and `interesting`, it would be relatively easy to figure out the correct label for `that`.

Now, we show how bidirectional inference works on this sample. Suppose we use beam search with width of 2, and we use a window of (-2, 2) for context features.

For the first step, we enumerate hypotheses for each word. For example, `found` could have a label VBN or VBD. Suppose that at this point the most favorable action, out of the candidate hypotheses, is the assignment of NN to `book`, according to the context features defined on words. Then, we resolve the label for `book` first. We maintain the top two hypotheses as shown below. Here, the second most favorable label for `book` is VB.

```
                    NN
                    VB
Agatha found that book interesting
  w1     w2    w3   w4      w5
             (Step 1)
```

At the second step, assume the most favorable action is the assignment of label JJ to `interesting` in the context of NN for `book`. Then we maintain the top two hypotheses for span `book interesting` as shown below. The second most favorable label for `interesting` is still JJ, but in the context of VB for `book`.

```
                   NN------JJ
                   VB------JJ
Agatha found that book interesting
  w1     w2    w3   w4      w5
             (Step 2)
```

Then, suppose we are most confident for assigning labels VBD and VBN to `found`, in that order. We get two separated tagged spans as shown below.

```
       VBD         NN------JJ
       VBN         VB------JJ
Agatha found that book interesting
  w1     w2    w3   w4      w5
             (Step 3)
```

In the next step, suppose we are most confident for assigning label DT to `that` under the context of VBD on the left and NN-JJ on the right side, as shown below (second most favorable action, not discussed here, is also displayed). After tagging w_3, two separated spans merge into one, starting from `found` to `interesting`.

```
       VBD---DT---NN------JJ
       VBD---IN---NN------JJ
Agatha found that book interesting
  w1     w2    w3   w4      w5
             (Step 4)
```

For the last step, we assign label NNP to `Agatha`, which could be an out-of-vocabulary word, under the context of VBD-DT on the right.

```
  NNP---VBD---DT---NN------JJ
  NNP---VBD---IN---NN------JJ
Agatha found that book interesting
  w1     w2    w3   w4      w5
             (Step 5)
```

This simple example has shown the advantage of adopting a flexible search strategy. However, it is still unclear how we maintain the hypotheses, how we keep candidates and accepted labels and spans, and how we employ dynamic programming. We will answer these questions in the formal definition of the inference algorithm in the next section.

2.2 Inference Algorithm

Terminology: Let the input sequence be $w_1 w_2 \cdots w_n$. For each token w_i, we are expected to assign a label $t_i \in \mathbf{T}$, with $\mathbf{T}$ the label set.

A subsequence $w_i \cdots w_j$ is called a **span**, and is denoted $[i, j]$. Each span p considered by the algorithm is associated with one or more **hypotheses**, that is, sequences over $\mathbf{T}$ having the same length as p. Part of the label sequence of each hypothesis is used as a context for labeling tokens outside the span p. For example, if a tri-gram model is adopted, we use the two labels on the left boundary and the two labels on the right boundary of the hypothesis for labeling outside tokens. The left two labels are called the **left interface**, and the right two labels are called the **right interface**. Left and right interfaces have only one label in case of spans of length one.

A pair $s = (I_{left}, I_{right})$ with a left and a right interface is called a **state**. We partition the hypotheses associated with span p into sets compatible with the same state. In practice, for span p, we use a matrix M_p indexed by states, so that $M_p(s)$, $s = (I_{left}, I_{right})$, is the set of all hypotheses associated with p that are compatible with I_{left} and I_{right}.

For a span p and a state s, we denote the associated top hypothesis as

$$s.T = \operatorname*{argmax}_{h \in M_p(s)} V(h),$$

where V is the score of a hypothesis (defined in (1) below). Similarly, we denote the top state for p as

$$p.S = \operatorname*{argmax}_{s\colon M_p(s) \neq \emptyset} V(s.T).$$

Therefore, for each span p, we have a top hypothesis $p.S.T$, whose score is the highest among all the hypotheses for span p.

Hypotheses are started and grown by means of labeling actions. For each hypothesis h associated with a span p we maintain its most recent labeling action $h.A$, involving some token within p, as well as the states $h.S_L$ and $h.S_R$ that have been used as context by such an action, if any. Note that $h.S_L$ and $h.S_R$ refer to spans that are subsequences of p. We recursively compute the score of h as

$$V(h) = V(h.S_L.T) + V(h.S_R.T) + U(h.A), \quad (1)$$

Algorithm 1 Inference Algorithm

Require: token sequence $w_1 \cdots w_n$;
Require: beam width B;
Require: weight vector $\mathbf{w}$;

```
Initialize P, the set of accepted spans;
Initialize Q, the queue of candidate spans;
repeat
    span p' ← argmax_{p∈Q} U(p.S.T.A);
    Update P with p';
    Update Q with p' and P;
until (Q = ∅)
```

where U is the score of an action. In other words, the score of an hypothesis is the sum of the score of the most recent action $h.A$ and the scores of the top hypotheses of the context states. The score of an action $h.A$ is computed through a linear function whose weight vector is $\mathbf{w}$, as

$$U(h.A) = \mathbf{w} \cdot \mathbf{f}(h.A), \quad (2)$$

where $\mathbf{f}(h.A)$ is the feature vector of action $h.A$, which depends on $h.S_L$ and $h.S_R$.

Algorithm: Algorithm 1 is the inference algorithm. We are given the input sequence and two parameters, beam width B to determine the number of states maintained for each span, and weight vector $\mathbf{w}$ used to compute the score of an action.

We first initialize the set P of accepted spans with the empty set. Then we initialize the queue Q of candidate spans with span $[i, i]$ for each token w_i, and for each $t \in \mathbf{T}$ assigned to w_i we set

$$M_{[i,i]}((t, t)) = \{i \rightarrow t\},$$

where $i \rightarrow t$ represents the hypothesis consisting of a single action which assigns label t to w_i. This provides the set of starting hypotheses.

As for the example `Agatha found that book interesting` in the previous subsection, we have

- $P = \emptyset$
- $Q = \{[1, 1], [2, 2], [3, 3], [4, 4], [5, 5]\}$

Suppose NN and VB are the two possible POS tags for w_4 `book`. We have

- $M_{[4,4]}(\text{NN}, \text{NN}) = \{h_{441} = 4 \rightarrow \text{NN}\}$
- $M_{[4,4]}(\text{VB}, \text{VB}) = \{h_{442} = 4 \rightarrow \text{VB}\}$

The most recent action of hypothesis h_{441} is to assign NN to w_4. According to Equation (2), the score of this action $U(h_{441}.A)$ depends on the features defined on the local context of action. For example,

$$f_{1001}(h_{441}.A) = \begin{cases} 1 & \text{if } t = \text{NN} \wedge w^{-1} = \texttt{that} \\ 0 & \text{otherwise,} \end{cases}$$

where w^{-1} represents the left word. It should be noted that, for all the features depending on the neighboring tags, the value is always 0, since those tags are still unknown in the step of initialization. Since this operation does not depend on solved tags, we have $V(h_{441}) = U(h_{411}.A)$, according to Equation (1).

The core of the algorithm repeatedly selects a candidate span from Q, and uses it to update P and Q, until a span covering the whole sequence is added to P and Q becomes empty. This is explained in detail below.

At each step, we remove from Q the span p' such that the *action* (not hypothesis) score of its top hypothesis, $p'.S.T$, is the highest. This represents the labeling action for the next move that we are most confident about. Now we need to update P and Q with the selected span p'. We add p' to P, and remove from P the spans included in p', if any. Let $\mathcal{S}$ be the set of removed spans. We remove from Q each span which takes one of the spans in $\mathcal{S}$ as context, and replace it with a new candidate span taking p' (and another accepted span) as context. We always maintain B different states for each span.

Back to the previous example, after Step 3 is completed, w_2 `found`, w_4 `book` and w_5 `interesting` have been tagged and we have

- $P = \{[2,2],[4,5]\}$
- $Q = \{[1,2],[2,5]\}$

There are two candidate spans in Q, each with its associated hypotheses and most recent actions. More specifically, we can either solve w_1 based on the context hypotheses for $[2,2]$, resulting in span $[1,2]$, or else solve w_3 based on the context hypotheses in $[2,2]$ and $[4,5]$, resulting in span $[2,5]$.

The top two states for span $[2,2]$ are

- $M_{[2,2]}(\text{VBD, VBD}) = \{h_{221} = 2 \rightarrow \text{VBD}\}$
- $M_{[2,2]}(\text{VBN, VBN}) = \{h_{222} = 2 \rightarrow \text{VBN}\}$

and the top two states for span $[4,5]$ are

- $M_{[4,5]}(\text{NN-JJ, NN-JJ})$ $= \{h_{451} = (\text{NN,NN})5 \rightarrow \text{JJ}\}$
- $M_{[4,5]}(\text{VB-JJ, VB-JJ})$ $= \{h_{452} = (\text{VB,VB})5 \rightarrow \text{JJ}\}$

Here $(\text{NN,NN})5 \rightarrow \text{JJ}$ represents the hypothesis coming from the action of assigning JJ to w_5 under the left context state of (NN,NN). $(\text{VB,VB})5 \rightarrow \text{JJ}$ has a similar meaning.[1]

We first compute the hypotheses resulting from all possible POS tag assignments to w_3, under all possible state combinations of the neighboring spans $[2,2]$ and $[4,5]$. Suppose the highest score action consists in the assignment of DT under the left context state (VBD, VBD) and the right context state (NN-JJ, NN-JJ). We obtain hypothesis $h_{251} = (\text{VBD,VBD})3 \rightarrow \text{DT(NN-JJ, NN-JJ)}$ with

$$\begin{aligned} V(h_{251}) &= V((\text{VBD,VBD}).T) + \\ &\quad V((\text{NN-JJ,NN-JJ}).T) + U(h_{251}.A) \\ &= V(h_{221}) + V(h_{451}) + \mathbf{w} \cdot \mathbf{f}(h_{251}.A) \end{aligned}$$

Here, features for action $h_{251}.A$ may depend on the left tag VBD and right tags NN-JJ, which have been solved before. More details of the feature functions are given in Section 4.2. For example, we can have features like

$$f_{2002}(h_{251}.A) = \begin{cases} 1 & \text{if } t = \text{DT} \wedge t^{+2} = \text{JJ} \\ 0 & \text{otherwise,} \end{cases}$$

We maintain the top two states with the highest hypothesis scores, if the beam width is set to two. We have

- $M_{[2,5]}(\text{VBD-DT, NN-JJ}) = \{h_{251} = (\text{VBD,VBD})3 \rightarrow \text{DT(NN-JJ,NN-JJ)}\}$
- $M_{[2,5]}(\text{VBD-IN, NN-JJ}) = \{h_{252} = (\text{VBD,VBD})3 \rightarrow \text{IN(NN-JJ,NN-JJ)}\}$

Similarly, we compute the top hypotheses and states for span $[1,2]$. Suppose now the hypothesis with the highest *action* score is h_{251}. Then we update P by adding $[2,5]$ and removing $[2,2]$ and $[4,5]$, which are covered by $[2,5]$. We also update Q by removing $[2,5]$ and $[1,2]$,[2] and add new candidate span $[1,5]$ resulting in

- $P = \{[2,5]\}$
- $Q = \{[1,5]\}$

[1]It should be noted that, in these cases, each state contains only one hypothesis. However, if the span is longer than 4 words, there may exist multiple hypotheses for the same state. For example, hypotheses DT-NN-VBD-DT-JJ and DT-NN-VBN-DT-JJ have the same left interface DT-NN and right interface DT-JJ.

[2]Span $[1,2]$ depends on $[2,2]$ and $[2,2]$ has been removed from P. So it is no longer a valid candidate given the accepted spans in P.

The algorithm is especially designed in such a way that, at each step, some new span is added to P or else some spans already present in P are extended by some token(s). Furthermore, no pair of overlapping spans is ever found in P, and the number of pairs of overlapping spans that may be found in Q is always bounded by a constant. This means that the algorithm performs at most n iterations, and its running time is therefore $\mathcal{O}(B^2 n)$, that is, linear in the length of the input sequence.

2.3 Learning Algorithm

In this section, we propose *guided learning*, a Perceptron like algorithm, to learn the weight vector $\mathbf{w}$, as shown in Algorithm 2. We use $p'.G$ to represent the gold standard hypothesis on span p'.

For each input sequence X_r and the gold standard sequence of labeling Y_r, we first initialize P and Q as in the inference algorithm. Then we select the span for the next move as in Algorithm 1. If $p'.S.T$, the top hypothesis of the selected span p', is compatible with the gold standard, we update P and Q as in Algorithm 1. Otherwise, we update the weight vector in the Perceptron style, by promoting the features of the gold standard action, and demoting the features of the action of the top hypothesis. Then we re-generate the queue Q with P and the updated weight vector $\mathbf{w}$. Specifically, we first remove all the elements in Q, and then generate hypotheses for all the possible spans based on the context spans in P. Hypothesis scores and action scores are calculated with the updated weight vector $\mathbf{w}$.

A special aspect of Algorithm 2 is that we maintain two scores: the score of the action represents the confidence for the next move, and the score of the hypothesis represents the overall quality of a partial result. The selection for the next action directly depends on the score of the action, but not on the score of the hypothesis. On the other hand, the score of the hypothesis is used to maintain top partial results for each span.

We briefly describe the soundness of the Guided Learning Algorithm in terms of two aspects. First, in Algorithm 2 weight update is activated whenever there exists an incorrect state s, the action score of whose top hypothesis $s.T$ is higher than that of any state in each span. We demote this action and promote the gold standard action on the same span.

Algorithm 2 Guided Learning Algorithm

Require: training sequence pairs $\{(X_r, Y_r)\}_{1\le r\le R}$;
Require: beam width B and iterations I;
1: $\mathbf{w} \leftarrow \mathbf{0}$;
2: **for** $(i \leftarrow 1; i \le I; i{+}{+})$ **do**
3:   **for** $(r \leftarrow 1; r \le R; r{+}{+})$ **do**
4:     Load sequence X_r and gold labeling Y_r.
5:     Initialize P, the set of accepted spans
6:     Initialize Q, the queue of candidate spans;
7:     **repeat**
8:       $p' \leftarrow \mathrm{argmax}_{p\in Q}\, U(p.S.T.A)$;
9:       **if** $(p'.S.T = p'.G)$ **then**
10:         Update P with p';
11:         Update Q with p' and P;
12:       **else**
13:         $\mathit{promote}(\mathbf{w}, \mathbf{f}(p'.G.A))$;
14:         $\mathit{demote}(\mathbf{w}, \mathbf{f}(p'.S.T.A))$;
15:         Re-generate Q with $\mathbf{w}$ and P;
16:       **end if**
17:     **until** $(Q = \emptyset)$
18:   **end for**
19: **end for**

However, we do not automatically adopt the gold standard action on this span. Instead, in the next step, the top hypothesis of another span might be selected based on the score of action, which means that it becomes the most favorable action according to the updated weights.

As a second aspect, if the action score of a gold standard hypothesis is higher than that of any others, this hypothesis and the corresponding span are guaranteed to be selected at line 8 of Algorithm 2. The reason for this is that the scores of the context hypotheses of a gold standard hypothesis must be no less than those of other hypotheses of the same span. This could be shown recursively with respect to Equation 1, because the context hypotheses of a gold standard hypothesis are also compatible with the gold standard.

Furthermore, if we take

$$(\mathbf{x_i} = \mathbf{f}(p'.G.A) - \mathbf{f}(p'.S.T.A), y_i = +1)$$

as a positive sample, and

$$(\mathbf{x_j} = \mathbf{f}(p'.S.T.A) - \mathbf{f}(p'.G.A), y_j = -1)$$

as a negative sample, the weight updates at lines 13 and 14 are a stochastic approximation of gradient descent that minimizes the squared errors of the misclassified samples (Widrow and Hoff, 1960). What is special with our learning algorithm is the strategy used to select samples for training.

In general, this novel learning framework lies between supervised learning and reinforcement learning. Guided learning is more difficult than supervised learning, because we do not know the order of inference. The order is learned automatically, and partial output is in turn used to train the local classifier. Therefore, the order of inference and the local classification are dynamically incorporated in the learning phase.

Guided learning is not as hard as reinforcement learning. At each local step in learning, we always know the undesirable labeling actions according to the gold standard, although we do not know which is the most desirable. In this approach, we can easily collect the automatically generated negative samples, and use them in learning. These negative samples are exactly those we will face during inference with the current weight vector.

In our experiments, we have used Averaged Perceptron (Collins, 2002; Freund and Schapire, 1999) and Perceptron with margin (Krauth and Mézard, 1987) to improve performance.

3 Related Works

Tsuruoka and Tsujii (2005) proposed a bidirectional POS tagger, in which the order of inference is handled with the easiest-first heuristic. Giménez and Màrquez (2004) combined the results of a left-to-right scan and a right-to-left scan. In our model, the order of inference is dynamically incorporated into the training of the local classifier.

Toutanova et al. (2003) reported a POS tagger based on cyclic dependency network. In their work, the order of inference is fixed as from left to right. In this approach, large beam width is required to maintain the ambiguous hypotheses. In our approach, we can handle tokens that we are most confident about first, so that our system does not need a large beam. As shown in Section 4.2, even deterministic inference shows rather good results.

Our guided learning can be modeled as a search algorithm with Perceptron like learning (Daumé III and Marcu, 2005). However, as far as we know, the mechanism of bidirectional search with an online learning algorithm has not been investigated before. In (Daumé III and Marcu, 2005), as well as other similar works (Collins, 2002; Collins and Roark, 2004; Shen and Joshi, 2005), only left-to-right search was employed. Our guided learning algorithm provides more flexibility in search with an automatically learned order. In addition, our treatment of the score of action and the score of hypothesis is unique (see discussion in Section 2.3).

Data Set	Sections	Sentences	Tokens
Training	0-18	38,219	912,344
Develop	19-21	5,527	131,768
Test	22-24	5,462	129,654

Table 1: Data set splits

Furthermore, compared to the above works, our guided learning algorithm is more aggressive on learning. In (Collins and Roark, 2004; Shen and Joshi, 2005), a search stops if there is no hypothesis compatible with the gold standard in the queue of candidates. In (Daumé III and Marcu, 2005), the search is resumed after some gold standard compatible hypotheses are inserted into a queue for future expansion, and the weights are updated correspondingly. However, there is no guarantee that the updated weights assign a higher score to those inserted gold standard compatible hypotheses. In our algorithm, the gold standard compatible hypotheses are used for weight update only. As a result, after each sentence is processed, the weight vector can usually successfully predict the gold standard parse. Therefore our learning algorithm is *aggressive* on weight update.

As far as this aspect is concerned, our algorithm is similar to the MIRA algorithm in (Crammer and Singer, 2003). In MIRA, one always knows the correct hypothesis. In our case, we do not know the correct order of operations. So we use our form of weight update to implement aggressive learning.

4 Experiments on POS Tagging

4.1 Settings

We apply our guided learning algorithm to POS tagging. We carry out experiments on the standard data set of the Penn Treebank (PTB) (Marcus et al., 1994). Following (Ratnaparkhi, 1996; Collins, 2002; Toutanova et al., 2003; Tsuruoka and Tsujii, 2005),

Feature Sets	Templates	Error%
A	Ratnaparkhi's	3.05
B	A + $[t_0, t_1]$, $[t_0, t_{-1}, t_1]$, $[t_0, t_1, t_2]$	2.92
C	B + $[t_0, t_{-2}]$, $[t_0, t_2]$, $[t_0, t_{-2}, w_0]$, $[t_0, t_{-1}, w_0]$, $[t_0, t_1, w_0]$, $[t_0, t_2, w_0]$, $[t_0, t_{-2}, t_{-1}, w_0]$, $[t_0, t_{-1}, t_1, w_0]$, $[t_0, t_1, t_2, w_0]$	2.84
D	C + $[t_0, w_{-1}, w_0]$, $[t_0, w_1, w_0]$	2.78
E	D + $[t_0, X = \text{prefix or suffix of } w_0], 4 < \vert X \vert \leq 9$	2.72

Table 2: Experiments on the *development* data with beam width of 3

we cut the PTB into the training, development and test sets as shown in Table 1. We use tools provided by CoNLL-2005 [3] to extract POS tags from the *mrg* files of PTB. So the data set is the same as previous work. We use the development set to select features and estimate the number of iterations in training. In our experiments, we enumerate all the POS tags for each word instead of using a dictionary as in (Ratnaparkhi, 1996), since the size of the tag set is tractable and our learning algorithm is efficient enough.

4.2 Results

Effect of Features: We first run the experiments to evaluate the effect of features. We use templates to define features. For this set of experiments, we set the beam width $B = 3$ as a balance between speed and accuracy. The guided learning algorithm usually converges on the development data set in 4-8 iterations over the training data.

Table 2 shows the error rate on the development set with different features. We first use the same feature set used in (Ratnaparkhi, 1996), which includes a set of prefix, suffix and lexical features, as well as some bi-gram and tri-gram context features. Following (Collins, 2002), we do not distinguish rare words. On set A, Ratnaparkhi's feature set, our system reports an error rate of 3.05% on the development data set.

With set B, we include a few feature templates which are symmetric to those in Ratnaparkhi's set, but are only available with bidirectional search. With set C, we add more bi-gram and tri-gram features. With set D, we include bi-lexical features. With set E, we use prefixes and suffixes of length up to 9, as in (Toutanova et al., 2003; Tsuruoka and Tsujii, 2005). We obtain 2.72% of error rate. We will use this feature set on our final experiments on the test data.

Effect of Search and Learning Strategies: For the second set of experiments, we evaluate the effect of search methods, learning strategies, and beam width. We use feature set E for this set of experiments. Table 3 shows the error rates on the development data set with both left-to-right (L-to-R) and bidirectional (Bi-Dir) search methods. We also tested both aggressive learning and non-aggressive learning strategies with beam width of 1 and 3.

Search	Aggressive?	Beam=1	Beam=3
L-to-R	Yes	2.94	2.82
L-to-R	No	3.24	2.75
Bi-Dir	Yes	2.84	2.72
Bi-Dir	No	does not converge	

Table 3: Experiments on the *development* data

First, with non-aggressive learning on bidirectional search, the error rate does not converge to a comparable number. This is due to the fact that the search space is too large in bidirectional search, if we do not use aggressive learning to constrain the samples for learning.

With aggressive learning, the bidirectional approach always shows advantages over left-to-right search. However, the gap is not large. This is due to the fact that the accuracy of POS tagging is very high. As a result, we can always keep the gold-standard tags in the beam even with left-to-right search in training.

This can also explain why the performance of left-to-right search with non-aggressive learning is close to bidirectional search if the beam is large enough. However, with beam width = 1, non-aggressive learning over left-to-right search performs much worse, because in this case it is more likely that the gold-standard tag is not in the beam.

This set of experiments show that guided learning is more preferable for tasks with higher ambiguities. In our recent work (Shen and Joshi, 2007), we have applied a variant of this algorithm to dependency parsing, and showed significant improvement over left-to-right non-aggressive learning strategy.

Comparison: Table 4 shows the comparison with the previous works on the PTB test sections.

[3] http://www.lsi.upc.es/˜srlconll/soft.html, package srlconll-1.1.tgz.

System	Beam	Error%
(Ratnaparkhi, 1996)	5	3.37
(Tsuruoka and Tsujii, 2005)	1	2.90
(Collins, 2002)	-	2.89
Guided Learning, feature B	3	2.85
(Tsuruoka and Tsujii, 2005)	all	2.85
(Giménez and Màrquez, 2004)	-	2.84
(Toutanova et al., 2003)	-	2.76
Guided Learning, feature E	1	2.73
Guided Learning, feature E	3	2.67

Table 4: Comparison with the previous works

According to the experiments shown above, we build our best system by using feature set E with beam width $B = 3$. The number of iterations on the training data is estimated with respect to the development data. We obtain an error rate of **2.67%** on the test data. With deterministic search, or beam with $B = 1$, we obtain an error rate of 2.73%.

Compared to previous best result on the same data set, 2.76% by (Toutanova et al., 2003), our best result shows a relative error reduction of 3.3%. This result is very promising, since we have not used any specially designed features in our experiments. It is reported in (Toutanova et al., 2003) that a crude company name detector was used to generate features, and it gave rise to significant improvement in performance. However, it is difficult for us to duplicate exactly the same feature for the purpose of comparison, although it is convenient to use features like that in our framework.

5 Conclusions

In this paper, we propose *guided learning*, a new learning framework for bidirectional sequence classification. The tasks of learning the order of inference and training the local classifier are dynamically incorporated into a single Perceptron like algorithm.

We apply this novel algorithm to POS tagging. It obtains an error rate of 2.67% on the standard PTB test set, which represents 3.3% relative error reduction over the previous best result (Toutanova et al., 2003) on the same data set, while using fewer features. By using deterministic search, it obtains an error rate of 2.73%, a 5.9% relative error reduction over the previous best deterministic algorithm (Tsuruoka and Tsujii, 2005). It should be noted that the error rate is close to the inter-annotator discrepancy on PTB, the standard test set for POS tagging, therefore it is very difficult to achieve improvement.

References

L. Bottou. 1991. *Une approche théorique de l'apprentissage connexionniste: Applications à la reconnaissance de la parole*. Ph.D. thesis, Université de Paris XI.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL-2004*.

M. Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP-2002*.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

H. Daumé III and D. Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML-2005*.

Y. Freund and R. E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

J. Giménez and L. Màrquez. 2004. Svmtool: A general pos tagger generator based on support vector machines. In *LREC-2004*.

W. Krauth and M. Mézard. 1987. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20:745–752.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmentation and labeling sequence data. In *ICML-2001*.

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

A. Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *EMNLP-1996*.

G. Satta and O. Stock. 1994. Bi-Directional Context-Free Grammar Parsing for Natural Language Processing. *Artificial Intelligence*, 69(1-2).

L. Shen and A. K. Joshi. 2005. Incremental LTAG Parsing. In *EMNLP-2005*.

L. Shen and A. K. Joshi. 2007. Bidirectional LTAG Dependency Parsing. Technical Report 07-02, IRCS, UPenn.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin markov networks. In *NIPS-2003*.

K. Toutanova, D. Klein, C. Manning, and Y. Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL-2003*.

Y. Tsuruoka and J. Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *EMNLP-2005*.

B. Widrow and M. E. Hoff. 1960. Adaptive switching circuits. *IRE WESCON Convention Record, part 4*.

W. Woods. 1976. Parsers in speech understanding systems. Technical Report 3438, Vol. 4, 1–21, BBN Inc.