

Semantic Role Recognition using Kernels on Weighted Marked Ordered Labeled Trees

Jun'ichi Kazama and Kentaro Torisawa

Japan Advanced Institute of Science and Technology (JAIST)

Asahidai 1-1, Nomi, Ishikawa, 923-1292 Japan

{kazama, torisawa}@jaist.ac.jp

Abstract

We present a method for recognizing semantic role arguments using a kernel on weighted marked ordered labeled trees (the WMOLT kernel). We extend the kernels on marked ordered labeled trees (Kazama and Torisawa, 2005) so that the mark can be weighted according to its importance. We improve the accuracy by giving more weights on subtrees that contain the predicate and the argument nodes with this ability. Although Kazama and Torisawa (2005) presented fast training with tree kernels, the slow classification during runtime remained to be solved. In this paper, we give a solution that uses an efficient DP updating procedure applicable in argument recognition. We demonstrate that the WMOLT kernel improves the accuracy, and our speed-up method makes the recognition more than 40 times faster than the naive classification.

1 Introduction

Semantic role labeling (SRL) is a task that recognizes the arguments of a predicate (verb) in a sentence and assigns the correct role to each argument. As this task is recognized as an important step after (or the last step of) syntactic analysis, many studies have been conducted to achieve accurate semantic role labeling (Gildea and Jurafsky, 2002; Moschitti, 2004; Hacioglu et al., 2004; Punyakanok et al., 2004; Pradhan et al., 2005a; Pradhan et al., 2005b; Toutanova et al., 2005).

Most of the studies have focused on machine learning because of the availability of standard datasets, such as PropBank (Kingsbury and Palmer, 2002). Naturally, the usefulness of parse trees in

this task can be anticipated. For example, the recent CoNLL 2005 shared task (Carreras and Màrquez, 2005) provided parse trees for use and their usefulness was ensured. Most of the methods heuristically extract features from parse trees, and from other sources, and use them in machine learning methods based on feature vector representation. As a result, these methods depend on feature engineering, which is time-consuming.

Tree kernels (Collins and Duffy, 2001; Kashima and Koyanagi, 2002) have been proposed to directly handle trees in kernel-based methods, such as SVMs (Vapnik, 1995). Tree kernels calculate the similarity between trees, taking into consideration all of the subtrees, and, therefore there is no need for such feature engineering.

Moschitti and Bejan (2004) extensively studied tree kernels for semantic role labeling. However, they reported that they could not successfully build an accurate argument recognizer, although the role assignment was improved. Although Moschitti et al. (2005) reported on argument recognition using tree kernels, it was a preliminary evaluation because they used oracle parse trees.

Kazama and Torisawa (2005) proposed a new tree kernel for node relation labeling, as which SRL can be cast. This kernel is defined on marked ordered labeled trees, where a node can have a mark to indicate the existence of a relation. We refer to this kernel as the MOLT kernel. Compared to (Moschitti and Bejan, 2004) where tree fragments are heuristically extracted before applying tree kernels, the MOLT kernel is general and desirable since it does not require such fragment extraction. However, the evaluation conducted by Kazama and Torisawa (2005) was limited to preliminary experiments for role assignment. In this study, we first evaluated the performance of the MOLT kernel for argument recognition, and found that the MOLT kernel cannot achieve a high accuracy if used in its original form.

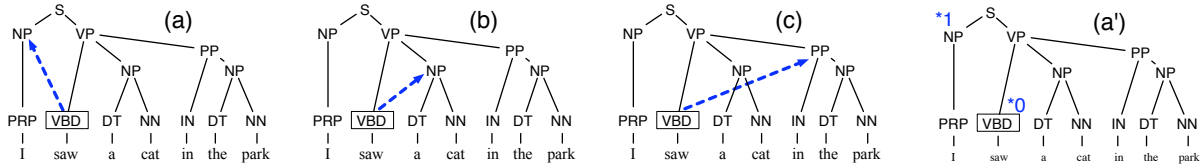


Figure 1: (a)-(c): Argument recognition as node relation recognition. (a’): relation (a) represented as marked ordered tree.

Therefore, in this paper we propose a modification of the MOLT kernel, which greatly improves the accuracy. The problem with the original MOLT kernel is that it treats subtrees with one mark, i.e., those including only the argument or the predicate node, and subtrees with two marks, i.e., those including both the argument and the predicate nodes equally, although the latter is likely to be more important for distinguishing difficult arguments. Thus, we modified the MOLT kernel so that the marks can be weighted in order to give large weights to the subtrees with many marks. We call the modified kernel the WMOLT kernel (the kernel on *weighted marked ordered labeled trees*). We show that this modification greatly improves the accuracy when the weights for marks are properly tuned.

One of the issues that arises when using tree kernels is time complexity. In general, tree kernels can be calculated in $O(|T_1||T_2|)$ time, where $|T_i|$ is the number of nodes in tree T_i , using dynamic programming (DP) procedures (Collins and Duffy, 2001; Kashima and Koyanagi, 2002). However, this cost is not negligible in practice. Kazama and Torisawa (2005) proposed a method that drastically speeds up the calculation during training by converting trees into efficient vectors using a tree mining algorithm. However, the slow classification during runtime remained an open problem.

We propose a method for speeding up the runtime classification for argument recognition. In argument recognition, we determine whether a node is an argument or not for all the nodes in a tree. This requires a series of calculations between a support vector tree and a tree with slightly different marking. By exploiting this property, we can efficiently update DP cells to obtain the kernel value with less computational cost.

In the experiments, we demonstrated that the WMOLT kernel drastically improved the accuracy

and that our speed-up method enabled more than 40 times faster argument recognition. Despite these successes, the performance of our current system is $F_1 = 78.22$ on the CoNLL 2005 evaluation set when using the Charniak parse trees, which is far worse than the state-of-the-art system. We will present possible reasons and future directions.

2 Semantic Role Labeling

Semantic role labeling (SRL) recognizes the arguments of a given predicate and assigns the correct role to each argument. For example, the sentence “I saw a cat in the park” will be labeled as follows with respect to the predicate “see”.

[A0 I] [V saw] [A1 a cat] [AM-LOC in the park]

In the example, A0, A1, and AM-LOC are the roles assigned to the arguments. In the CoNLL 2005 dataset, there are the numbered arguments (A-X) whose semantics are predicate dependent, the adjuncts (AM-X), and the references (R-X) for relative clauses.

Many previous studies employed two-step SRL methods, where (1) we first recognize the arguments, and then (2) classify the argument to the correct role. We also assume this two-step processing and focus on the argument recognition.

Given a parse tree, argument recognition can be cast as the classification of tree nodes into two classes, “ARG” and “NO-ARG”. Then, we consider the words (a phrase) that are the descendants of an “ARG” node to be an argument. Since arguments are defined for a given predicate, this classification is the recognition of a relation between the predicate and tree nodes. Thus, we want to build a binary classifier that returns a +1 for correct relations and a -1 for incorrect relations. For the above example, the classifier will output a +1 for the relations indicated by (a), (b), and (c) in Figure 1 and a -1 for the relations between the predicate node and other nodes.

Since the task is the classification of trees with node relations, tree kernels for usual ordered labeled trees, such as those proposed by Collins and Duffy (2001) and Kashima and Koyanagi (2002), are not useful. Kazama and Torisawa (2005) proposed to represent a node relation in a tree as a marked ordered labeled tree and presented a kernel for it (MOLT kernel). We adopted the MOLT kernel and extend it for accurate argument recognition.

3 Kernels for Argument Recognition

3.1 Kernel-based classification

Kernel-based methods, such as support vector machines (SVMs) (Vapnik, 1995), consider a mapping $\Phi(x)$ that maps the object x into a, (usually high-dimensional), feature space and learn a classifier in this space. A kernel function $K(x_i, x_j)$ is a function that calculates the inner product $\langle \Phi(x_i), \Phi(x_j) \rangle$ in the feature space without explicitly computing $\Phi(x)$, which is sometimes intractable. Then, any classifier that is represented by using only the inner products between the vectors in a feature space can be rewritten using the kernel function. For example, an SVM classifier has the form:

$$f(x) = \sum_i \alpha_i K(x_i, x) + b,$$

where α_i and b are the parameters learned in the training. With kernel-based methods, we can construct a powerful classifier in a high-dimensional feature space. In addition, objects x do not need to be vectors as long as a kernel function is defined (e.g., x can be strings, trees, or graphs).

3.2 MOLT kernel

A marked ordered labeled tree (Kazama and Torisawa, 2005) is an ordered labeled tree in which each node can have a mark in addition to a label. We can encode a k -node relation by using k distinct marks. In this study, we determine an argument node without considering other arguments of the same predicate, i.e., we represent an argument relation as a two-node relation using two marks. For example, the relation (a) in Figure 1 can be represented as the marked ordered labeled tree (a').¹

¹Note that we use mark *0 for the predicate node and mark *1 for the argument node.

Table 1: Notations for MOLT kernel.

- n_i denotes a node of a tree. In this paper, n_i is an ID assigned in the post-order traversal.
- $|T_i|$ denotes the number of nodes in tree T_i .
- $l(n_i)$ returns the label of node n_i .
- $m(n_i)$ returns the mark of node n_i . If n_i has no mark, $m(n_i)$ returns the special mark no-mark.
- $marked(n_i)$ returns *true* iff $m(n_i)$ is not no-mark.
- $nc(n_i)$ is the number of children of node n_i .
- $ch_k(n_i)$ is the k -th child of node n_i .
- $pa(n_i)$ is the parent of node n_i .
- $root(T_i)$ is the root node of T_i
- $n_i \succeq n_j$ means that n_i is an elder sister of n_j .

Kazama and Torisawa (2005) presented a kernel on marked ordered trees (the MOLT kernel), which is defined as:²

$$K(T_1, T_2) = \sum_{i=1}^E W(S_i) \cdot \#_{S_i}(T_1) \cdot \#_{S_i}(T_2),$$

where S_i is a possible subtree and $\#_{S_i}(T_j)$ is the number of times S_i is included in T_j . The mapping corresponding to this kernel is $\Phi(T) = (\sqrt{W(S_1)}\#_{S_1}(T), \dots, \sqrt{W(S_E)}\#_{S_E}(T))$, which maps the tree into the feature space of all the possible subtrees.

The tree inclusion is defined in many ways. For example, Kashima and Koyanagi (2002) presented the following type of inclusion.








1 DEFINITION S is included in T iff there exists a one-to-one function ψ from a node of S to a node of T , such that (i) $pa(\psi(n_i)) = \psi(pa(n_i))$, (ii) $\psi(n_i) \succeq \psi(n_j)$ iff $n_i \succeq n_j$, and (iii) $l(\psi(n_i)) = l(n_i)$ (and $m(\psi(n_i)) = m(n_i)$ in the MOLT kernel).

See Table 1 for the meaning of each function. This definition means that any subtrees preserving the parent-child relation, the sibling relation, and label-marks, are allowed. In this paper, we employ this definition, since Kazama and Torisawa (2005) reported that the MOLT kernel with this definition has a higher accuracy than one with the definition presented by Collins and Duffy (2001).

$W(S_i)$ is the weight of subtree S_i . The weighting in Kazama and Torisawa (2005) is written as fol-

²This notation is slightly different from (Kazama and Torisawa, 2005).

Table 2: Example of subtree inclusion and subtree weights. The last row shows the weights for WMOLT kernel.

T	included subtrees					
						
$W(S_i)$	0	λ	λ	λ^2	λ^2	λ^3
$W(S_i)$	0	$\lambda\gamma$	$\lambda\gamma$	$\lambda^2\gamma$	$\lambda^2\gamma^2$	$\lambda^3\gamma^2$

lows.

$$W(S_i) = \begin{cases} \lambda^{|S_i|} & \text{if } \text{marked}(S_i), \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where $\text{marked}(S_i)$ returns *true* iff $\text{marked}(n_i) = \text{true}$ for at least one node in tree S_i . By this weighting, only the subtrees with at least one mark are considered. The idea behind this is that subtrees having no marks are not useful for relation recognition or labeling. λ ($0 \leq \lambda \leq 1$) is a factor to prevent the kernel values from becoming too large, which has been used in previous studies (Collins and Duffy, 2001; Kashima and Koyanagi, 2002).

Table 2 shows an example of subtree inclusion and the weights given to each included subtree. Note that the subtrees are treated differently when the markings are different, even if the labels are the same.

Although the dimension of the feature space is exponential, tree kernels can be calculated in $O(|T_1||T_2|)$ time using dynamic programming (DP) procedures (Collins and Duffy, 2001; Kashima and Koyanagi, 2002). The MOLT kernel also has an $O(|T_1||T_2|)$ DP procedure (Kazama and Torisawa, 2005).

3.3 WMOLT kernel

Although Kazama and Torisawa (2005) evaluated the MOLT kernel for SRL, the evaluation was only on the role assignment task and was preliminary. We evaluated the MOLT kernel for argument recognition, and found that the MOLT kernel cannot achieve a high accuracy for argument recognition.

The problem is that the MOLT kernel treats subtrees with one mark and subtrees with two marks equally, although the latter seems to be more important in distinguishing difficult arguments.

Consider the sentence, “He said industry should

build plants”. For “say”, we have the following labeling.

[A0 He] [V said] [A1 industry should build plants]

On the other hand, for “build”, we have

He said [A0 industry] [AM-MOD should] [V build]
[A1 plants].

As can be seen, “he” is the A0 argument of “say”, but not an argument of “build”. Thus, our classifier should return a +1 for the tree where “he” is marked when the predicate is “say”, and a -1 when the predicate is “build”. Although the subtrees around the node for “say” and “build” are different, the subtrees around the node for “he” are identical for both cases. If “he” is often the A0 argument in the corpus, it is likely that the classifier returns a +1 even for “build”. Although the subtrees containing both the predicate and the argument nodes are considered in the MOLT kernel, they are given relatively small weights by Eq. (1), since such subtrees are large.

Thus, we modify the MOLT kernel so that the mark can be weighted according to its importance and the more marks the subtrees contain, the more weights they get. The modification is simple. We change the definition of $W(S_i)$ as follows.

$$W(S_i) = \begin{cases} \lambda^{|S_i|} \prod_{n_i \in S_i} \gamma(m(n_i)) & \text{if } \text{marked}(S_i), \\ 0 & \text{otherwise,} \end{cases}$$

where $\gamma(m)$ (≥ 1) is the weight of mark m . We call a kernel with this weight the WMOLT kernel. In this study, we assume $\gamma(\text{no-mark}) = 1$ and $\gamma(*0) = \gamma(*1) = \gamma$. Then, the weight is simplified as follows.

$$W(S_i) = \begin{cases} \lambda^{|S_i|} \gamma^{\#_m(S_i)} & \text{if } \text{marked}(S_i), \\ 0 & \text{otherwise,} \end{cases}$$

where $\#_m(S_i)$ is the number of marked nodes in S_i . The last row in Table 2 shows how the subtree weights change by introducing this mark weighting.

For the WMOLT kernel, we can derive $O(|T_1||T_2|)$ DP procedure by slightly modifying the procedure presented by Kazama and Torisawa (2005). The method for speeding up training described in Kazama and Torisawa (2005) can also be applied with a slight modification.

Algorithm 3.1: WMOLT-KERNEL(T_1, T_2)

```

for  $n_1 \leftarrow 1$  to  $|T_1|$  do // nodes are ordered by the post-order traversal
   $m \leftarrow \text{marked}(n_1)$ 
  for  $n_2 \leftarrow 1$  to  $|T_2|$  do // actually iterate only on  $n_2$  with  $l(n_1) = l(n_2)$ 
    if  $l(n_1) \neq l(n_2)$  or  $m(n_1) \neq m(n_2)$  then
       $C(n_1, n_2) \leftarrow 0$   $C^r(n_1, n_2) \leftarrow 0$ 
    else if  $n_1$  and  $n_2$  are leaf nodes then
      if  $m$  then  $C(n_1, n_2) \leftarrow \lambda \cdot \gamma$ ;  $C^r(n_1, n_2) \leftarrow \lambda \cdot \gamma$  else  $C(n_1, n_2) \leftarrow \lambda$ ;  $C^r(n_1, n_2) \leftarrow 0$ 
    else
       $S(0, j) \leftarrow 1$ ,  $S(i, 0) \leftarrow 1$  ( $i \in [0, nc(n_1)], j \in [0, nc(n_2)]$ )
      if  $m$  then  $S^r(0, j) \leftarrow 1$ ,  $S^r(i, 0) \leftarrow 1$  else  $S^r(0, j) \leftarrow 0$ ,  $S^r(i, 0) \leftarrow 0$ 
      for  $i \leftarrow 1$  to  $nc(n_1)$  do
        for  $j \leftarrow 1$  to  $nc(n_2)$  do
           $S(i, j) \leftarrow S(i-1, j) + S(i, j-1) - S(i-1, j-1) + S(i-1, j-1) \cdot C(ch_i(n_1), ch_j(n_2))$ 
           $S^r(i, j) \leftarrow S^r(i-1, j) + S^r(i, j-1) - S^r(i-1, j-1) + S^r(i-1, j-1) \cdot C(ch_i(n_1), ch_j(n_2))$ 
             $+ S(i-1, j-1) \cdot C^r(ch_i(n_1), ch_j(n_2)) - S^r(i-1, j-1) \cdot C^r(ch_i(n_1), ch_j(n_2))$ 
        if  $m$  then  $C(n_1, n_2) \leftarrow \lambda \cdot \gamma \cdot S(nc(n_1), nc(n_2))$  else  $C(n_1, n_2) \leftarrow \lambda \cdot S(nc(n_1), nc(n_2))$ 
        if  $m$  then  $C^r(n_1, n_2) \leftarrow \lambda \cdot \gamma \cdot S^r(nc(n_1), nc(n_2))$  else  $C^r(n_1, n_2) \leftarrow \lambda \cdot S^r(nc(n_1), nc(n_2))$ 
    return  $(\sum_{n_1=1}^{|T_1|} \sum_{n_2=1}^{|T_2|} C^r(n_1, n_2))$ 

```

We describe this DP procedure in some detail. The key is the use of two DP matrices of size $|T_1| \times |T_2|$. The first is $C(n_1, n_2)$ defined as:

$$C(n_1, n_2) \equiv \sum_{S_i} W'(S_i) \cdot \#_{S_i}(T_1 \Delta n_1) \cdot \#_{S_i}(T_2 \Delta n_2),$$

where $\#_{S_i}(T_j \Delta n_k)$ represents the number of times subtree S_i is included in tree T_j with $\psi(\text{root}(S_i)) = n_k$. $W'(S_i)$ is defined as $W'(S_i) = \lambda^{|S_i|} \gamma^{\#m(S_i)}$. This means that this matrix records the values that ignore whether $\text{marked}(S_i) = \text{true}$ or not. The second is $C^r(n_1, n_2)$ defined as:

$$C^r(n_1, n_2) \equiv \sum_{S_i} W(S_i) \cdot \#_{S_i}(T_1 \Delta n_1) \cdot \#_{S_i}(T_2 \Delta n_2).$$

With these matrices, the kernel is calculated as:

$$K(T_1, T_2) = \sum_{n_1 \in T_1} \sum_{n_2 \in T_2} C^r(n_1, n_2).$$

$C(n_1, n_2)$ and $C^r(n_1, n_2)$ are calculated recursively, starting from the leaves of the trees. The recursive procedure is shown in Algorithm 3.1. See also Table 1 for the meaning of the functions used.

4 Fast Argument Recognition

We use the SVMs for the classifiers in argument recognition in this study and describe the fast classification method based on SVMs.³ We denote a marked ordered labeled tree where node n_k of an ordered labeled tree U is marked by mark X , n_l by Y , and so on, by $U@ \{n_k = X, n_l = Y, \dots\}$.

³The method can be applied to a wide range of kernel-based methods that have the same structure as SVMs.

Algorithm 4.1: CALCULATE-T(U, T_j)

```

procedure FAST-UPDATE( $n_k$ )
   $diff \leftarrow 0$ ,  $m(n_k) \leftarrow *1$ ,  $\mathcal{U} \leftarrow \phi$ 
  for  $n_2 \leftarrow 1$  to  $|T_j|$  do  $\text{change}(n_2) \leftarrow \text{true}$ 
   $n_1 \leftarrow n_k$ 
  while  $n_1 \neq \text{nil}$  do
    for  $n_2 \leftarrow 1$  to  $|T_j|$  do
      // actually iterate only on  $n_2$  with  $l(\text{pa}(n_1)) = l(n_2)$ 
       $nchange(n_2) \leftarrow \text{false}$ 
      for  $n_2 \leftarrow 1$  to  $|T_j|$  do
        // actually iterate only on  $n_2$  with  $l(n_1) = l(n_2)$ 
        if  $\text{change}(n_2)$  then
           $pre \leftarrow C^r(n_1, n_2)$ ,  $\mathcal{U} \leftarrow \mathcal{U} \cup (n_1, n_2)$ 
          update  $C(n_1, n_2)$  and  $C^r(n_1, n_2)$ 
            using (A) of Algorithm 3.1
           $diff += (C^r(n_1, n_2) - pre)$ 
          if  $\text{pa}(n_2) \neq \text{nil}$  then  $nchange(\text{pa}(n_2)) \leftarrow \text{true}$ 
           $n_1 \leftarrow \text{pa}(n_1)$ ,  $\text{change} \leftarrow nchange$ 
    for  $(n_1, n_2) \in \mathcal{U}$  do //restore DP cells
       $C(n_1, n_2) \leftarrow C'(n_1, n_2)$ ,  $C^r(n_1, n_2) \leftarrow C^{r'}(n_1, n_2)$ 
     $m(n_k) \leftarrow \text{no-mark}$ 
  return ( $diff$ )

main
   $m(n_v) \leftarrow *0$ ,  $k \leftarrow \text{WMOLT-KERNEL}(U, T_j)$ 
   $C'(n_1, n_2) \leftarrow C(n_1, n_2)$ ,  $C^{r'}(n_1, n_2) \leftarrow C^r(n_1, n_2)$ 
  for  $n_k \leftarrow 1$  to  $|U|$  do ( $n_k \neq n_v$ )
     $diff \leftarrow \text{FAST-UPDATE}(n_k)$ ,  $t(n_k) \leftarrow k + diff$ 

```

Given a sentence represented by tree U and the node for the target predicate n_v , the argument recognition requires the calculation of:

$$s(n_k) = \sum_{T_j \in \mathcal{SV}} \alpha_j K(U@ \{n_v = *0, n_k = *1\}, T_j) + b, \quad (2)$$

for all $n_k \in U$ ($\neq n_v$), where \mathcal{SV} represents the support vectors. Naively, this requires $O(|U| \times |\mathcal{SV}| \times |U| |T_j|)$ time, which is rather costly in practice.

However, if we exploit the fact that $U@ \{n_v = *0, n_k = *1\}$ is different from $U@ \{n_v = *0\}$ at one node, we can greatly speed up the above calculation. At first, we calculate $K(U@ \{n_v = *0\}, T_j)$ using the DP procedure presented in the previous section, and then calculate $K(U@ \{n_v = *0, n_k = *1\}, T_j)$ using a more efficient DP that updates only the values of the necessary DP cells of the first DP. More specifically, we only need to update the DP cells involving the ancestor nodes of n_k .

Here we show the procedure for calculating $t(n_k) = K(U@ \{n_v = *0, n_k = *1\}, T_j)$ for all n_k for a given support vector T_j , which will suffice for calculating $s(n_k)$. Algorithm 4.1 shows the procedure. For each n_k , this procedure updates at most $(n_k$'s depth) $\times |T_j|$ cells, which is much less than $|U| \times |T_j|$ cells. In addition, when updating the cells for (n_1, n_2) , we only need to update them when the cells for any child of n_2 have been updated in the calculation of the cells for the children of n_1 . To achieve this, *change*(n_2) in the algorithm stores whether the cells of any child of n_2 have been updated. This technique will also reduce the number of updated cells.

5 Non-overlapping Constraint

Finally, in argument recognition, there is a strong constraint that the arguments for a given predicate do not overlap each other. To enforce this constraint, we employ the approach presented by Toutanova et al. (2005). Given the local classification probability $p(n_k = X_k)$ ($X_k \in \{\text{ARG}, \text{NO-ARG}\}$), this method finds the assignment that maximizes $\prod_k p(n_k = X_k)$ while satisfying the above non-overlapping constraint, by using a dynamic programming procedure. Since the output of SVMs is not a probability value, in this study we obtain the probability value by converting the output from the SVM, $s(n_k)$, using the sigmoid function:⁴

$$p(n_k = \text{ARG}) = 1/(1 + \exp(-s(n_k))).$$

6 Evaluation

6.1 Setting

For our evaluation we used the dataset provided for the CoNLL 2005 SRL shared task

⁴Parameter fitting (Platt, 1999) is not performed.

(www.lsi.upc.edu/~srlconll). We used only the training part and divided it into our training, development, and test sets (23,899, 7,966, and 7,967 sentences, respectively). We used the outputs of the Charniak parser provided with the dataset. We also used POS tags, which were also provided, by inserting the nodes labeled by POS tags above the word nodes. The words were downcased.

We used TinySVM⁵ as the implementation of the SVMs, adding the WMOLT kernel. We normalized the kernel as: $K(T_i, T_j) / \sqrt{K(T_i, T_i) \times K(T_j, T_j)}$.

To train the classifiers, for a positive example we used the marked ordered labeled tree that encodes an argument in the training set. Although nodes other than the argument nodes were potentially negative examples, we used 1/5 of these nodes that were randomly-sampled, since the number of such nodes is so large that the training cannot be performed in practice. Note that we ignored the arguments that do not match any node in the tree (the rate of such arguments was about 3.5% in the training set).

6.2 Effect of mark weighting

We first evaluated the effect of the mark weighting of the WMOLT kernel. For several fixed γ , we tuned λ and the soft-margin constant of the SVM, C , and evaluated the recognition accuracy. We tested 30 different values of $C \in [0.1 \dots 500]$ for each $\lambda \in [0.05, 0.1, 0.15, 0.2, 0.25, 0.3]$. The tuning was performed using the method for speeding up the training with tree kernels described by Kazama and Torisawa (2005). We conducted the above experiment for several training sizes.

Table 3 shows the results. This table shows the best setting of λ and C , the performance on the development set with the best setting, and the performance on the test set. The performance is shown in the F_1 measure. Note that we treated the region labeled C- k in the CoNLL 2005 dataset as an independent argument.

We can see that the mark weighting greatly improves the accuracy over the original MOLT kernel (i.e., $\gamma = 1$). In addition, we can see that the best setting for γ is somewhere around $\gamma = 4,000$. In this experiment, we could only test up to 1,000 sentences due to the cost of SVM training, which were

⁵chasen.org/~taku/software/TinySVM

Table 3: Effect of γ in mark weighting of WMOLT kernel.

γ	training size (No. of sentences)											
	250			500			700			1,000		
	setting (λ, C)	dev (F_1)	test (F_1)	setting (λ, C)	dev (F_1)	test (F_1)	setting (λ, C)	dev (F_1)	test (F_1)	setting (λ, C)	dev (F_1)	test (F_1)
1	0.15, 20.50	63.66	65.13	0.2, 20.50	69.01	70.33	0.2, 20.50	72.11	73.57	0.25, 12.04	75.38	76.25
100	0.3, 12.04	80.13	80.85	0.3, 500	82.25	82.98	0.3, 34.92	83.93	84.72	0.3, 3.18	85.09	85.85
1,000	0.2, 2.438	82.65	83.36	0.2, 2.438	84.80	85.45	0.2, 3.182	85.58	86.20	0.2, 7.071	86.40	86.80
2,000	0.2, 2.438	83.43	84.12	0.2, 2.438	85.56	86.24	0.2, 2.438	86.23	86.80	0.2, 12.04	86.61	87.18
4,000	0.2, 2.438	83.87	84.50	0.15, 4.15	84.94	85.61	0.15, 7.07	85.84	86.32	0.2, 12.04	86.82	87.31
4,000 (w/o)		80.81	81.41		80.71	81.51		81.86	82.33		84.27	84.63

empirically $O(L^2)$ where L is the number of training examples, regardless of the use of the speed-up method (Kazama and Torisawa, 2005). However, we can observe that the WMOLT kernel achieves a high accuracy even when the training data is very small.

6.3 Effect of non-overlapping constraint

Additionally, we observed how the accuracy changes when we do not use the method described in Section 5 and instead consider the node to be an argument when $s(n_k) > 0$. The last row in Table 3 shows the accuracy for the model obtained with $\gamma = 4,000$. We could observe that the non-overlapping constraint also improves the accuracy.

6.4 Recognition speed-up

Next, we examined the method for fast argument recognition described in Section 4. Using the classifiers with $\gamma = 4,000$, we measured the time required for recognizing the arguments for 200 sentences with the naive classification of Eq. (2) and with the fast update procedure shown in Algorithm 4.1. The time was measured using a computer with 2.2-GHz dual-core Opterons and 8-GB of RAM.

Table 4 shows the results. We can see a constant speed-up by a factor of more than 40, although the time was increased for both methods as the size of the training data increases (due to the increase in the number of support vectors).

Table 4: Recognition time (sec.) with naive classification and proposed fast update.

	training size (No. of sentences)			
	250	500	750	1,000
naive	11,266	13,008	18,313	30,226
proposed	226	310	442	731
speed-up	49.84	41.96	41.43	41.34

6.5 Evaluation on CoNLL 2005 evaluation set

To compare the performance of our system with other systems, we conducted the evaluation on the official evaluation set of the CoNLL 2005 shared task. We used a model trained using 2,000 sentences (57,547 examples) with ($\gamma = 4,000, \lambda = 0.2, C = 12.04$), the best setting in the previous experiments. This is the largest model we have successfully trained so far, and has $F_1 = 88.00$ on the test set in the previous experiments.

The accuracy of this model on the official evaluation set was $F_1 = 79.96$ using the criterion from the previous experiments where we treated a C- k argument as an independent argument. The official evaluation script returned $F_1 = 78.22$. This difference is caused because the official script takes C- k arguments into consideration, while our system cannot output C- k labels since it is just an argument recognizer. Therefore, the performance will become slightly higher than $F_1 = 78.22$ if we perform the role assignment step. However, our current system is worse than the systems reported in the CoNLL 2005 shared task in any case, since it is reported that they had $F_1 = 79.92$ to 83.78 argument recognition accuracy (Carreras and Màrquez, 2005).

7 Discussion

Although we have improved the accuracy by introducing the WMOLT kernel, the accuracy for the official evaluation set was not satisfactory. One possible reason is the accuracy of the parser. Since the Charniak parser is trained on the same set with the training set of the CoNLL 2005 shared task, the parsing accuracy is worse for the official evaluation set than for the training set. For example, the rate of the arguments that do not match any node of the parse tree is 3.93% for the training set, but 8.16% for the

evaluation set. This, to some extent, explains why our system, which achieved $F_1 = 88.00$ for our test set, could only achieved $F_1 = 79.96$. To achieve a higher accuracy, we need to make the system more robust to parsing errors. Some of the non-matching arguments are caused by incorrect treatment of quotation marks and commas. These errors seem to be solved by using simple pre-processing. Other major non-matching arguments are caused by PP attachment errors. To solve these errors, we need to explore more, such as using n -best parses and the use of several syntactic views (Pradhan et al., 2005b).

Another reason for the low accuracy is the size of the training data. In this study, we could train the SVM with 2,000 sentences (this took more than 30 hours including the conversion of trees), but this is a very small fraction of the entire training set. We need to explore the methods for incorporating a large training set within a reasonable training time. For example, the combination of small SVMs (Shen et al., 2003) is a possible direction.

The contribution of this study is not the accuracy achieved. The first contribution is the demonstration of the drastic effect of the mark weighting. We will explore more accurate kernels based on the WMOLT kernel. For example, we are planning to use different weights depending on the marks. The second contribution is the method of speeding-up argument recognition. This is of great importance, since the proposed method can be applied to other tasks where all nodes in a tree should be classified. In addition, this method became possible because of the WMOLT kernel, and it is hard to apply to Moschitti and Bejan (2004) where the tree structure changes during recognition. Thus, the architecture that uses the WMOLT kernel is promising, if we assume further progress is possible with the kernel design.

8 Conclusion

We proposed a method for recognizing semantic role arguments using the WMOLT kernel. The mark weighting introduced in the WMOLT kernel greatly improved the accuracy. In addition, we presented a method for speeding up the recognition, which resulted in more than a 40 times faster recognition. Although the accuracy of the current system is worse than the state-of-the-art system, we expect to further improve our system.

References

- X. Carreras and L. Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *CoNLL 2005*.
- M. Collins and N. Duffy. 2001. Convolution kernels for natural language. In *NIPS 2001*.
- D. Gildea and D. Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3).
- K. Hacioglu, S. Pradhan, W. Ward, J. H. Martin, and D. Jurafsky. 2004. Semantic role labeling by tagging syntactic chunks. In *CoNLL 2004*.
- H. Kashima and T. Koyanagi. 2002. Kernels for semi-structured data. In *ICML 2002*, pages 291–298.
- J. Kazama and K. Torisawa. 2005. Speeding up training with tree kernels for node relation labeling. In *EMNLP 2005*.
- P. Kingsbury and M. Palmer. 2002. From treebank to propbank. In *LREC 02*.
- A. Moschitti and C. A. Bejan. 2004. A semantic kernels for predicate argument classification. In *CoNLL 2004*.
- A. Moschitti, B. Coppola, D. Pighin, and B. Basili. 2005. Engineering of syntactic features for shallow semantic parsing. In *ACL 2005 Workshop on Feature Engineering for Machine Learning in Natural Language Processing*.
- A. Moschitti. 2004. A study on convolution kernels for shallow semantic parsing. In *ACL 2004*.
- J. C. Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*.
- S. Pradhan, K. Hacioglu, W. Ward, D. Jurafsky, and J. H. Martin. 2005a. Support vector learning for semantic argument classification. *Machine Learning*, 60(1).
- S. Pradhan, W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky. 2005b. Semantic role labeling using different syntactic views. In *ACL 2005*.
- V. Punyakanok, D. Roth, W. Yih, and D. Zimak. 2004. Semantic role labeling via integer linear programming inference. In *COLING 2004*.
- L. Shen, A. Sarkar, and A. K. Joshi. 2003. Using LTAG based features in parse reranking. In *EMNLP 2003*.
- K. Toutanova, A. Haghghi, and C. D. Manning. 2005. Joint learning improves semantic role labeling. In *ACL 2005*.
- V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer Verlag.