A Tree Transducer Model for Grammatical Error Correction

Jan Buys and Brink van der Merwe
MIH Media Lab and Computer Science Division
Stellenbosch University, South Africa
janbuys@ml.sun.ac.za, abvdm@cs.sun.ac.za

Abstract

We present an approach to grammatical error correction for the CoNLL 2013 shared task based on a weighted tree-to-string transducer. Rules for the transducer are extracted from the NUCLE training data. An n-gram language model is used to rerank k-best sentence lists generated by the transducer. Our system obtains a precision, recall and F1 score of 0.27, 0.1333 and 0.1785, respectively, on the official test set. On the revised annotations, the F1 score increases to 0.2505. Our system ranked 6th out of the participating teams on both the original and revised test set annotations.

1 Introduction

There has recently been an increase in research on automated grammatical error detection and correction for writing by English language learners (Leacock et al., 2010). In the most prominent line of research, statistical classifiers are trained to detect or correct specific error types. Features for these classifiers are based on word context and local syntactic information. The classifiers are combined, and a language model is often used to filter corrections. Research on this approach focusses especially on preposition and determiner errors. Most of the systems in the HOO 2011 and 2012 shared tasks (Dale and Kilgarriff, 2011; Dale et al., 2012) fall under this broad approach.

In a second class of models, a model for generating corrected sentences is formulated in the noisy-channel framework, relying strongly on a language model to distinguish between grammatical and ungrammatical candidate corrections (Lee and Seneff, 2006; Turner and Charniak, 2007; Park and Levy, 2011). Such models are often inspired by techniques developed for statistical machine translation (Brockett et al., 2006). Finally, rule-based methods are often used in commercial language processing systems such as word processors. Here large hand-crafted linguistically expressive, error-tolerant grammars are used to analyse sentences and identify where constraints have been broken.

In this paper we present our system for the CoNLL 2013 shared task in grammatical error correction (Ng et al., 2013). Our grammar correction model is based on a tree-to-string transducer that is specified by a set of rules that each rewrite a tree fragment to a string of words and variables. These rules are extracted automatically from a set of training examples. Each training example consists of an incorrect sentence, a corresponding correct sentence with its parse tree, and a word alignment between the incorrect and correct sentences. During decoding the model searches for parsed well-formed sentences that could be transformed into a given incorrect sentence with high probability. Sentences are split into linguistically plausible clauses to decrease the average sentence length, in order to improve decoding runtime. In order to discriminate more accurately between candidate sentence corrections an n-gram language model trained on a large corpus of well-formed text is used to rerank the k-best hypotheses that the transducer model generates. The tree transducer and language model scores are weighted to maximize the model F1 score on a validation set. After decoding the clauses are recombined into the original sentence structure.

The next section describes preprocessing and the resources used by our system. Section 3 defines weighted tree-to-string transducers. We present the formulation of our error correction model in section 4, and discuss decoding with it in section 5. Section 6 describes language model reranking. System results are given in section 7. Finally, section 8 draws some conclusions and discuss future work.

Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pages 43–51,
Sofia, Bulgaria, August 8-9 2013.

2 Data Pre-processing

2.1 NUCLE

We use the pre-processed version of the NUCLE corpus (Dahlmeier et al., 2013) released as training data for this shared task. The data consist of essays, subdivided into paragraphs. Using NLTK (Bird et al., 2009), paragraphs were split into sentences with NLTK *punkt* and sentences were tokenized with NLTK *word tokenize*. Though this sentence-splitting and tokenization is not error-free (for example, quotation marks are handled incorrectly in some contexts), we use it to maintain consistency in our model. An error annotation in the data consists of the start and end token offsets in a sentence, as well as the correction that should replace the text between the offsets.

We divide the corpus into 80% training data, 10% validation data and 10% development data. Splitting is performed by random selection at essay level. For each sentence with corrections, we refer to the original as the incorrect sentence, and to the version with the corrections applied to it as the correct sentence. For the purpose of training our models, all words are lowercased. As described below, we also construct an alignment between the words of each of these sentence pairs.

The 2013 shared task focusses on five error types: Article or determiner, preposition, noun number, verb form, and subject-verb agreement errors. In the training data we only apply corrections of these types to obtain the correct version of the sentences, though other error types are also included in the error annotations. An alternative would be to apply the corrections of other error types to the correct and incorrect versions of the sentences. However, we decided against that in order to keep the training data realistically close to the test data, which will also contain these other errors. We do, however, correct some of the mechanical errors, especially spelling errors, in the incorrect and correct versions of the training data, to reduce noise that these errors may introduce into the model.

In order to train a syntax-based model for grammar correction, the correct version of the sentences are parsed with the Berkeley parser (Petrov and Klein, 2007). The Berkeley parser is a state-of-the-art unlexicalized parser. Given that the correct side of our training data will still contain errors, it is unlikely that lexicalized parsing will be more accurate. Parser options are set to obtain left-binarized parse trees under Viterbi decoding.

2.2 Wikipedia language model

We train the n-gram language model used in our system on a large corpus of text extracted from the English Wikipedia. The April 2013 Wikipedia XML dump[1] is used. This is parsed with the *gwtwiki*[2] Wikipedia parser, and all sentences consisting of 6 or more words are extracted. These sentences are tokenized with NLTK and lowercased. The corpus has about 1 500 millions words. As vocabulary we use the 64 000 words with the highest frequency occurrence in the corpus. A 3-gram language model is trained from the corpus on this restricted vocabulary, to keep the size of the language model reasonable. The language model is trained and applied with the SRILM toolkit (Stolcke, 2002). Kneser-Ney smoothing is used to estimate the model weights.

2.3 Vocabulary

We set the vocabulary of the transducer model as the union of the vocabulary of our language model and the vocabulary of the words of the correct sentences in the NUCLE training data. In the transducer construction we ensure that all words in this vocabulary can be accepted. A large number of URLs occur in the training data, as citations are included in some of the essays. We replace these with a `<url>` symbol to reduce noise in the vocabulary.

In the validation, development and test data, words that do not appear in the vocabulary are replaced with an `<unk>` symbol. We record the replaced words, so that after decoding they can be replaced back to their original positions. We do not perform automatic spelling correction on the test data as a preprocessing step: The occurrence of out of vocabulary words is small enough that performing spelling correction will not have a significant impact on the performance of our system.

We use the NLTK interface to WordNet (Miller, 1995) to find pairs of singular and plural nouns and groups of verbs that have the same base form. All verbs and non-proper nouns that occur in the language model vocabulary are grouped like this. The groups are used to construct additional rules for noun number and verb form errors.

[1] `http://download.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2`

[2] `http://code.google.com/p/gwtwiki/`

3 Weighted Tree-to-string Transducers

Tree transducers are a class of automata-theoretic models that perform transformations on tree structures. There is a rich theory concerning these models, and tree transducers with different restrictions can compute different classes of transformations. Algorithms for weighted variants of these transducers have recently been developed (Graehl et al., 2008; May, 2010) and applied to syntax-based statistical machine translation.

Tree-to-string transducers are a class of tree transducers that generalizes synchronous context-free grammars. These transducers can be used to transform strings into trees: For a given output string, the decoding problem is to find the input tree that could be transformed into the given string with the highest probability. This decoding process is referred to as backward application. The formulation is due to the noisy-channel model often followed in statistical machine translation.

3.1 Definitions

We need a few preliminary definitions (the notation of May (2010) is generally followed): A *ranked alphabet* Σ is a finite set of symbols, each which can take a finite set of ranks. A tree $t \in T_\Sigma$ is denoted by $\sigma(t_1, \ldots, t_k)$, where $k \in \mathbb{N}$, $t_1, \ldots, t_k \in T_\Sigma$ and σ is a node of rank k. Σ_k denotes the subset of Σ of all symbols with *rank* k. $T_\Sigma(S)$ is the set of all trees in $T_{\Sigma \cup S}$ where symbols from S occur only at the leaves. We donate by $X = \{x_1, x_2, \ldots\}$ a set of *variables*, and $X_k = \{x_1, \ldots, x_k\}$. With respect to X_k, a tree $u \in T_\Sigma(X_k)$ or a sequence $u \in (\Delta \cup (Q \times X))^*$ is *linear* if each element of X_k occurs at most once in u, and *nondeleting* if each element of X_k occurs at least once in u.

Formally, a *weighted extended top-down tree-to-string transducer* M is a 5-tuple $(Q, \Sigma, \Delta, R, Q_d)$ (May, 2010, chap. 2). Q is an alphabet of states that all have rank one. Σ and Δ are the ranked input and output alphabets, respectively. Q_d is the set of initial states. R is a finite set of rules with an associated weight function $\pi : R \rightarrow \mathcal{W}$. Each rule $r \in R$ is of the form $q.t \rightarrow g$ for $q \in Q, t \in T_\Sigma(X)$ and $g \in (\Delta \cup (Q \times X))^*$. The tree t should be linear in X, and each variable in g should also be in t.

We refer to $q.t$ as the left hand side of a rule, and to g as the right hand side. M is *linear* if the right hand side of each rule is linear, and *nondeleting* if the right hand side of each rule is nondeleting with respect to X_k, the set of variables on the left hand side.

For a rule $r : q.t \rightarrow g$ and $e, f \in (\Delta \cup (Q \times T_\Sigma))^*$, a *derivation step* $e \Rightarrow^r f$ is obtained by replacing the left-most element of e of the form $q(s)$, where s matches t, by a transformation of g, where each instance of a variable is replaced with the corresponding subtree of s. The sequence $d = (r_1, \ldots, r_m)$ is a *derivation* of the pair (t, s) if $t \Rightarrow^{r_1} t_1 \Rightarrow^{r_2} \ldots \Rightarrow^{r_m} s$, where $t_i \in (\Delta \cup (Q \times T_\Sigma))^*$ for $1 \leq i < m$. The weight of d is $wt(d) = \pi(r_1) \cdot \ldots \cdot \pi(r_m)$.

The tree transducer that we use is a weighted linear, nondeleting top-down tree-to-string transducer. The expressive power of this transducer class is sufficient for the transformations that we need our model to perform. Relaxing these restrictions will increase the decoding complexity of the transducer model significantly. Since we work with binarized trees, no node will have a rank greater than 2. Our transducer only has one state, q. Look-ahead restrictions are added to restrict the variables on the left hand side to match specific constituents.

3.2 Probability model

A tree-to-string transducer can represent a conditional probability model for the output sentences given the input trees, or a joint probability model over the input trees and output strings. We will use a joint probability distribution for our model. Note that there is spurious ambiguity in the model at two levels: Firstly, it is possible that there can be different derivations for the same tree-string pair. However, during the application of the model this ambiguity occurs infrequently. Secondly, the model can generate different trees with the same yield.

Suppose c is a correct sentence in the set C of all possible correct sentences, and i is the given (possibly) incorrect sentence. Let $\tau(c)$ represent the set of all possible parse trees of c. Then we want to find sentence

$$\hat{c} = \arg\max_{c \in C} P(c, i) \quad (1)$$

$$= \arg\max_{c \in C} \Sigma_{\pi \in \tau(c)} P(\pi, i) \quad (2)$$

The rule probabilities for the joint model are conditioned on the root node of the rule left hand side (and not on the entire left hand side, as would

be the case for a conditional model). The model is trained from a set of derivations constructed from the training data, as will be described below. Let $f(r)$ be the number of times that rule r occurs in all the training derivations. Then the probability estimate of a rule is

$$p(r|\text{root}(r)) = \frac{f(r)}{\Sigma_{r':\text{root}(r')=\text{root}(r)} f(r')} \quad (3)$$

In the construction of the transducer, some rules that do not occur in the training derivations are added. In order to give non-zero weights to these added rules, we apply Good-Turing smoothing (Katz, 1987). This method has the advantage of decreasing the counts of low-frequency rules whose rule counts may provide unreliable probability estimates. For the rules of each of the root nonterminals, the counts of rules with frequencies between 0 and 5 are re-estimated.

4 Transducer Model Formulation

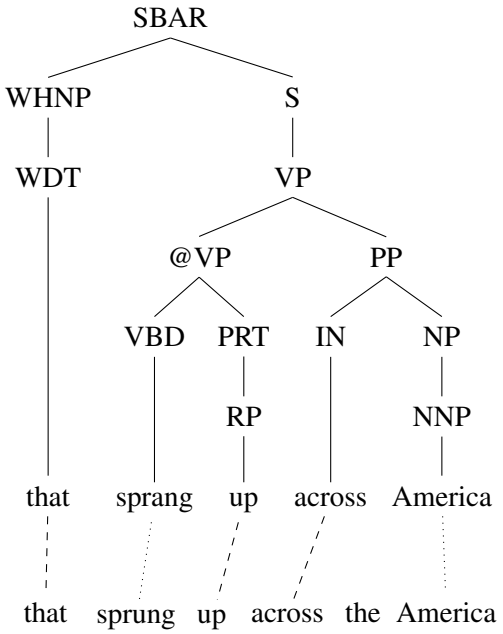
4.1 Word alignment

In order to extract rules from the training data to perform transformations between incorrect and correct sentences, we need to construct an alignment between words in each pair of correct and incorrect sentences. This approach is similar to aligning words in source and target language sentences for statistical machine translation. We construct the alignments from given sequences of edit operations in the training data. Figure 1(a) gives an example of a parse tree for a correct phrase, aligned with a corresponding incorrect phrase.

Firstly, all words in a sentence that do not occur in any edits are aligned one-to-one between the correct and incorrect sentence. In the example, "that", "up" and "across" are aligned in this way. Then, for each edit annotation, we consider the incorrect and correct phrases of that edit. Note that in the NUCLE annotations the incorrect phrase will always be non-empty, but the correct phrase may be empty.

Words that occur in both the correct and incorrect edit phrases are aligned one-to-one. We restrict such alignments to prevent overlapping. In the example in Figure 1(a), there is an edit to replace "the America" with "America", so the word "America" is aligned. Adding these alignments may split a phrase into unaligned subphrases. If such a subphrase is empty on either side, then the word(s) on the other side have to be left unaligned.

(a)



(b)

(1) q.WDT (*that*) → *that*
(2) q.VP ($x0$:@VP $x1$:PP) → $q.x0$ $q.x1$
(3) q.VBD (*sprang*) → *sprung*
(4) q.SBAR ($x0$:WHNP $x1$:S) → $q.x0$ $q.x1$
(5) q.PP ($x0$:IN $x1$:NP) → $q.x0$ *the* $q.x1$
(6) q.NNP (*America*) → *America*

Figure 1: (a) Example alignment between a correct parse tree and an incorrect clause. (b) Some rules extracted from the example.

In the example, "the" will be unaligned. But if the subphrases are non-empty on both sides, then all the words on the incorrect side are aligned to all the words on the correct side of the subphrase. In many instances, this will occur for single word replacements. In the example, "sprung" is aligned with "sprang" in this manner.

4.2 Rule extraction

We follow the GHKM transducer rule extraction algorithm described in (Galley et al., 2004) and (Galley et al., 2006). Given a training example (π, i, a), where π is the correct tree, i the incorrect sentence and a the alignment, rules are extracted for a tree-to-string derivation of (π, i) that is minimally consistent with the alignment a. Counts of how many times each rule is extracted over all the training examples are used to estimate the rule probabilities. A training example is represented as a directed graph as in Figure 1(a), with the edges

going downward.

For each of the nodes in π, we compute a *span* and a *complement span* with respect to the nodes in i. The span of a node n is defined by the indexes of the first and last words in f that are reachable from n. The spans of the leaves in the tree (the words of the correct sentence) are defined by a, and the spans of the other nodes can be computed bottom-up for each node from the spans of its child nodes. The complement span of n is the union of the spans of all nodes that are neither ancestors nor descendants of n. The complement spans can be computed top-down for each node by taking the union of the complement span of its parent and the spans of its siblings. Nodes whose spans and complement spans do not overlap, are called *frontier* nodes. From each frontier node, a rule can be extracted: The left hand side of the rule is a subtree rooted at n. The subtree is extracted by traversing π top-down from n, replacing all frontier nodes reached with variables (as more rules will be extracted from there). The right hand side is formed by the words of the span of n of the incorrect side, with the span of each left hand side frontier node replaced by the corresponding variable.

Figure 1(b) gives sample rules extracted from the training example in Figure 1(a). In the example tree, all the constituent nodes are frontier nodes, as there are no complex rewrites. For constituents under which no changes are made, CFG-like rules such as (1) and (2) are extracted. In the case where a single word is substituted (in the example, "sprang" with "sprung"), a rule for this substitution will be extracted (3). If there were no alignment between these words, the algorithm would have attached the word "sprung" to the rule headed by `SBAR` (4), which would clearly not have been linguistically sensible. In the case of the deletion of a word in the incorrect sentence, that word will left be unaligned ("the" in the example). The rule for this word (5) will have as head the lowest node that spans the words in i to the left and right of the unaligned word – in the example, the `PP` node. Rewrite rules for aligned words in phrase edits are also extracted (6).

4.3 Additional rules

We add rules to the transducer that involve words in the vocabulary. Whichever of these rules have not already been extracted from the training data will be assigned a rule count of 0, otherwise their rule counts will be left unchanged.

Non-lexicalized	
q.S ($x0$:NP $x1$:VP) → $q.x0$ $q.x1$	−0.596
q.S ($x0$:VP $x1$:VP) → $q.x0$ $q.x1$	−5.781
q.VP ($x0$:VP $x1$:SBAR) → $q.x0$ $q.x1$	−3.723
Word identity	
q.NN (*work*) → *work*	−2.614
q.VBP (*work*) → *work*	−2.475
q.DT (*the*) → *the*	−0.183
Single word substitution	
q.NN (*work*) → *works*	−4.343
q.VBP (*work*) → *working*	−4.541
q.VBZ (*works*) → *work*	−4.802
q.DT (*the*) → *a*	−3.100
q.IN (*of*) → *from*	−3.901
Phrase substitution	
q.NP (DT (*the*) NN (*right*)) → *rights*	−5.109
q.VP (VBG (*being*) VP (VBN (*researched*))) → *under researching*	−6.272
Context-sensitive phrase substitution	
q.VP (TO (*to*) VP (VB (*work*) $x0$:PP)) → *working* $q.x0$	−6.272
q.PP (IN (*in*) S (VP (VBG (*generating*) $x0$:NP))) → *to generate* $q.x0$	−5.480
Context-sensitive word insertion and deletion	
q.NP (DT (*the* $x0$:NN) → $q.x0$	−2.634
q.VP (VBZ (*has*) $x0$:VP) → $q.x0$	−5.202
q.VP ($x0$:VB $x1$:NP) → $q.x0$ *into* $q.x1$	−5.203

Table 1: Example transducer rules by type, with log probability weights.

We need to ensure that there are lexical rewrite rules for all the words in our vocabulary. For each of the words in the vocabulary we find one or two possible part-of-speech tags, using the NLTK POS tagger. We add word identity rules in the form of the examples in Table 1. An identity rule is also added for the `<unk>` symbol.

Additional rules are added for noun number and verb form errors, using the word groups extracted from the vocabulary and WordNet. These rules perform substitutions between singular and plural nouns (in both directions) and between verbs with the same base forms. Subject-verb agreement errors are also concerned with the verb form in the sentence, so added verb form rules will also be applicable to such errors. Examples of these single word substitutions are given in Table 1. Since determiner and preposition errors are restricted to a relatively small number of possible substitutions, we assume that all relevant rules involving these errors have already been extracted from the training data.

See Table 1 for rule examples categorized by the type of rewrite the rule performs. Examples of rules for all the error types under consideration are included. Log probability rule weights are also

given. Phrase substitution rules can be fully lexical (without variables) or context-sensitive (when they have variables). Word insertions and deletions will always be context-sensitive.

5 Transducer Model Decoding

5.1 Sentence to clause splitting

A challenge to our transducer model on the NUCLE dataset is the length of sentences. On the training data, 46% of sentences have length greater than 20 and 13% have length greater that 30. The decoding time of our model increases sharply when the length of sentences becomes greater than 20. For lengths greater that 30 decoding is not practically feasible on our available computational resources. In order to address this problem, we perform linguistically motivated sentence splits to decrease the length of sentences passed to the decoder. Clauses that are still longer than 30 words are not decoded. Decoding was performed on a desktop computer with 8GB RAM. To keep the overall decoding time reasonable, we restrict decoding to take no more than 1 minute per sentence on average.

Sentence splitting is based on constituency parses (obtained with the Berkeley parser) of the incorrect sentences under consideration. Sentences are split at clause level, using the heuristics described below. The goal is to extracted clauses that have a form similar to that of full sentences.

We distinguish between *S-clauses*, that are indicated by `S`, `SINV` and `SQ` parse tree constituents, and *SBAR-clauses*, indicated by `SBAR` or `SBARQ` parse tree constituents. An SBAR-clause usually consists of an introductory subordinating conjunction or *wh*-word, followed by an S-clause.

We perform splits on S-clauses. A clausal split is performed between the phrase before the start position of the S-clause and the phrase after that position. If the parse tree node of the S-clause is the child of an SBAR-clause node, the split is performed between the phrase before the starting position of the SBAR-clause, and the phrase after the start of the S-clause. The introductory words in the SBAR-clause are excluded from the extracted clauses.

Splits are also performed between some phrases separated by a coordinating conjunction, which is indicated by a `CC` tag in the parse tree. Such a split is performed only if the `CC` node is a child of an S-clause node. The phrase before the conjunction is split from the phrase after the conjunction, while the conjunction itself is excluded.

After decoding and reranking has been performed, the clauses are recombined to reconstruct the original sentences. For each clause the highest-scoring correct clause is chosen. Finally, the original case of all the words in the sentence is restored, as all words were lowercased in the model.

5.2 k-Best decoding

When performing decoding with the transducer model, we need to find the highest-scoring candidate correct sentences, so that we can in turn find the best sentence according to the overall model. We found that a good trade-off between speed and accuracy is to find a list of trees of the 1000-best derivations for a given (incorrect) sentence. The weights of different derivations for which the parse trees have the same yields, are summed to find weights for each of the hypothesis sentences. Note that this is an approximation of the summation in equation (2), which is taken over all parse trees with the same yield.

In our implementation the weighted tree transducer package Tiburon (May and Knight, 2006) is used. Tiburon implements generic operations on regular tree grammars, tree-to-tree and tree-to-string transducers. We use Tiburon to perform decoding in our model, using its implementation of backwards application and k-best decoding.

The decoding algorithm implemented by Tiburon is based on a weighted version of the Earley parsing algorithm (May, 2010, chap. 4). Empirically, large rules have a detrimental impact on the decoding speed of the algorithm. To address this problem, we extract rules from binarized parse trees, which results in smaller rules than using non-binarized parse trees. In Figure 1(a), the node `@VP` indicates that a binarization has been performed on the subtree `VP (VBD PRT PP)`. All remaining rules that have more that four variables are removed.

As the search space of the model is large, we need to apply some heuristic pruning. Following practices used in parsing models such as Huang and Chiang (2005), beam search is performed. The cell limit γ, the maximum number of hypotheses that can be kept at a state in the search process, is set to 30. The beam width β is set to 10^{-4}. This means that if a hypothesis score is worse than β times the score of the best partial hypothesis found

up to a specific point in the model, the hypothesis is discarded. γ and β were set to make decoding feasible on available computational resources.

The heuristic pruning may undermine some of the advantages our model might have in taking whole sentence analyses into account to generate error corrections. However, we find that despite this, the model is still able to generate hypothesis corrections that take non-local dependencies into consideration.

6 Language Model Reranking

Although the transducer model defines a joint probability distribution and is therefore sufficient to find corrections for given sentences, incorporating an n-gram language model in our system significantly increases its performance. The main reason for this is that the generative transducer model alone does not have enough discriminative power to distinguish between well-formed and ungrammatical sentences.

6.1 Evaluation

The standard evaluation metric used for grammatical error correction is precision, recall and F1 score. Changes made to a given incorrect sentence are represented by edits. For a sample sentence, the sufficient statistics for this evaluation metric is the 3-tuple (*#correct system edits*, *#system edits*, *#gold standard edits*). This can be summed over all the examples being evaluated, and the precision, recall and F1 scores can be computed from that.

The shared task uses the M^2 scorer, as described by Dahlmeier and Ng (2012). Given the original and system sentences, possible system edit sequences are represented with a lattice. The edit sequence that is the best match with the gold standard edit sequence is chosen to compute the edit scores.

6.2 Reranking

During decoding we compute the language model score for each of the hypothesis sentences generated by our transducer model for a given incorrect sentence. The log probability scores of the transducer and language models are normalized by the length of the incorrect sentence. In order to weigh these two scores, the transducer score is kept fixed, and the language model score is multiplied by a weight α. For a given incorrect sentence i and a generated set of hypothesis correct sentences $H(i)$, we want to find

$$\hat{c} = \arg\max_{c \in H(i)} [TT(c, i) + \alpha \cdot LM(c)] \quad (4)$$

where $TT(c, i)$ gives the tree transducer score and $LM(c)$ gives the language model score. The parameter α is set to maximize the F1 score of the model on a validation set. Let I be the set of incorrect sentences in this set. Then we want to find

$$\hat{\alpha} = \arg\max_{\alpha} \text{F1}[\Sigma_{i \in I} edits(\hat{c}, i, g(i))] \quad (5)$$

where $\hat{c}$ is given by (4) and *edits* is the sufficient statistics for the F1 score of $\hat{c}$ for the incorrect sentence i and gold standard edits $g(i)$.

Data set	Precision	Recall	F1 score
Validation	0.065	0.153	0.092
Development	0.079	0.149	0.103
Test (original)	0.2700	0.1333	0.1785
Test (revised)	0.3712	0.1891	0.2505

Table 2: Model results

7 Results

We now present results of the model on our validation and development sets, as well as on the official test set. A useful measure to analyze the performance of our model is to perform *oracle* reranking on the hypothesis sets generated by the transducer model. For each sentence, the oracle picks the hypothesis that will contribute to the best possible F1 score. We are especially interested in how frequently the correct sentence is among the hypothesis sentences – this is called the *hypothesis coverage*.

7.1 Development sets

On the development set, only 21% of clauses are annotated with corrections. For clauses that have no annotations, the hypothesis coverage is 99%, while for clauses that have annotations the hypothesis coverage is 49%. The oracle obtains a 0.64 F1 score.

We tune the value of α on the validation set to maximize the F1 score. The best F1 score is obtained with $\alpha = 1.6949$. The system results on the validation set and the development set with this α are given in Table 2. It was found that a strong

Error type	Development recall	Test recall
Noun number	0.2231	0.1818
Verb form	0.1839	0.1475
Article or determiner	0.1564	0.1261
Preposition	0.1655	0.0932
Subject-verb agreement	0.0957	0.1048

Table 3: Recall for each error type, on the development set and original test set.

weight on the language model (a relatively large α) increases the recall of the model.

A breakdown of the recall for each error type is given in Table 3. On the development set, the best recall is obtained for noun number errors, and the worst for subject-verb agreement errors. A reason for the relatively low performance on agreement errors may be due to the constituency parse tree representation used. In a clause, the subject noun phrase and the predicate verb phrase, whose head verb must agree with the subject, are in different subtrees. This increases the difficulty in modelling the dependency between the subject and the verb.

7.2 Test set

The test set released for this shared task consists of 1381 sentences, which we split into 2247 clauses using the heuristic described above. The distribution of sentence lengths is very similar to that of the training data. The number of out of vocabulary words is quite small at 0.03%. The set does not include any URLs, and the general impression was that it is less noisy than the training data.

The system result on the test set is given in Table 2. Scores for both the original and revised test data annotations are given. We submitted plausible corrections suggested by our system for the gold standard revision. This contributed to a significant increase in our model score on the revised annotations. The model recall on the test set is similar to that of the development on most error types, though the preposition error recall is significantly lower and the subject-verb agreement recall is slightly higher. This may indicate that preposition error correction rules in the model does not generalize well enough.

The precision of our model is significantly better on the test set than on the development set. This can be explained by differences in the characteristics of the test set. The relative occurrence of annotated errors is much higher in the test set than in the development set: 46% of clauses have corrections. It has been found previously that a low frequency of errors increase the difficulty of the correction task (Dahlmeier and Ng, 2011). This is caused especially by an increase in the number of system edits suggested for sentences that should not be changed. Our oracle found that for sentences that should not be changed, 100% of the correct unchanged hypotheses were generated by the tree transducer, while for sentences that should be changed, 50% of hypothesis sets contained the correct result. The oracle obtains a 0.75 F1 score. The precision of the oracle model increases significantly, from 0.65 to 0.95. Varying the choice of α controls the trade-off between precision and recall better on the test set than on the validation set. These results indicate that our model is more suited for data with the characteristics of the test set than for data similar to the development sets.

8 Conclusion

We presented a novel approach to grammatical error correction based on tree transducers, obtaining promising results. One of the weaknesses of our model is handling insertions and deletions. The model performs too many unnecessary deletions, especially removing content words or non-article determiners. It also has difficulty in finding edits where insertions such as article insertions should be performed.

For future work, ways of constructing better rule sets for the transducer should be investigated to take more dependencies into consideration and to improve probability estimates. Techniques to improve the runtime of the decoding algorithm while minimizing the loss in accuracy caused by heuristic pruning should be considered. Alternative approaches to reranking could also be investigated. Including additional features may increase the ability of the model to discriminate between grammatical and ungrammatical sentences.

Acknowledgement

The financial support of MIH is acknowledged.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

Chris Brockett, William B. Dolan, and Michael Gamon. 2006. Correcting ESL errors using phrasal SMT techniques. In *Proceedings of ACL*, pages 249–256.

Daniel Dahlmeier and Hwee Tou Ng. 2011. Grammatical error correction with alternating structure optimization. In *Proceedings of ACL*, pages 915–923.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of HTL-NAACL*, pages 568–572.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. 2013. Building a large annotated corpus of learner English: The NUS corpus of learner English. In *Proceedings of the 8th Workshop on Innovative Use of NLP for Building Educational Applications*, Atlanta, Georgia, USA.

Robert Dale and Adam Kilgarriff. 2011. Helping our own: The HOO 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 242–249, Nancy, France.

Robert Dale, Ilya Anisimoff, and George Narroway. 2012. HOO 2012: A report on the preposition and determiner error correction shared task. In *Proceedings of the 7th Workshop on the Innovative Use of NLP for Building Educational Applications*, pages 54–62, Montreal, Canada.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of HLT-NAACL*, pages 273–280.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of ACL*, pages 961–968.

Jonathan Graehl, Kevin Knight, and Jonathan May. 2008. Training tree transducers. *Computational Linguistics*, 34(3):391–427.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technologies*, pages 53–64.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.

Claudia Leacock, Martin Chodorow, Michael Gamon, and Joel R. Tetreault. 2010. *Automated Grammatical Error Detection for Language Learners*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

John Lee and Stephanie Seneff. 2006. Automatic grammar correction for second-language learners. In *Proceedings of Interspeech*, pages 1978–1981.

Jonathan May and Kevin Knight. 2006. Tiburon: A weighted tree automata toolkit. In *CIAA*, volume 4094 of *Lecture Notes in Computer Science*, pages 102–113. Springer.

Jonathan May. 2010. *Weighted Tree Automata and Transducers for Syntactic Natural Language Processing*. Ph.D. thesis, University of Southern California.

George A. Miller. 1995. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41.

Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. 2013. The CoNLL-2013 shared task on grammatical error correction. In *Proceedings of CoNLL*.

Y. Albert Park and Roger Levy. 2011. Automated whole sentence grammar correction using a noisy channel model. In *Proceedings of ACL*, pages 934–944.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT-NAACL*, pages 404–411.

Andreas Stolcke. 2002. SRILM: An extensible language modeling toolkit. In *Proceedings of ICSLP*, pages 901–904.

Jenine Turner and Eugene Charniak. 2007. Language modeling for determiner selection. In *Proceedings of HTL-NAACL*, pages 177–180.