A Beam-Search Decoder for Grammatical Error Correction

Daniel Dahlmeier[1] and **Hwee Tou Ng**[1,2]
[1]NUS Graduate School for Integrative Sciences and Engineering
[2]Department of Computer Science, National University of Singapore
{danielhe,nght}@comp.nus.edu.sg

Abstract

We present a novel beam-search decoder for grammatical error correction. The decoder iteratively generates new hypothesis corrections from current hypotheses and scores them based on features of grammatical correctness and fluency. These features include scores from discriminative classifiers for specific error categories, such as articles and prepositions. Unlike all previous approaches, our method is able to perform correction of whole sentences with multiple and interacting errors while still taking advantage of powerful existing classifier approaches. Our decoder achieves an F_1 correction score significantly higher than all previous published scores on the Helping Our Own (HOO) shared task data set.

1 Introduction

Grammatical error correction is an important problem in natural language processing (NLP) that has attracted an increasing amount of interest over the last few years. Grammatical error correction promises to provide instantaneous accurate feedback to language learners, e.g., learners of English as a Second Language (ESL).

The dominant paradigm that underlies most error correction systems to date is multi-class classification. A classifier is trained to predict a word from a *confusion set* of possible correction choices, given some feature representation of the surrounding sentence context. During testing, the classifier predicts the most likely correction for each test instance. If the prediction differs from the observed word used by the writer and the classifier is sufficiently confident in its prediction, the observed word is replaced by the prediction. Although considerable progress has been made, the classification approach suffers from some serious shortcomings. Each classifier corrects a single word for a specific error category individually. This ignores dependencies between the words in a sentence. Also, by conditioning on the surrounding context, the classifier implicitly assumes that the surrounding context is free of grammatical errors, which is often not the case. Finally, the classifier typically has to commit to a single one-best prediction and is not able to change its decision later or explore multiple corrections. Instead of correcting each word individually, we would like to perform global inference over corrections of whole sentences which can contain multiple and interacting errors.

An alternative paradigm is to view error correction as a statistical machine translation (SMT) problem from "bad" to "good" English. While this approach can naturally correct whole sentences, a standard SMT system cannot easily incorporate models for specific grammatical errors. It also suffers from the paucity of error-annotated training data for grammar correction. As a result, applying a standard SMT system to error correction does not produce good results, as we show in this work.

In this work, we present a novel beam-search decoder for grammatical error correction that combines the advantages of the classification approach and the SMT approach. Starting from the original input sentence, the decoder performs an iterative search over possible sentence-level hypotheses

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 568–578, Jeju Island, Korea, 12–14 July 2012.

to find the best sentence-level correction. In each iteration, a set of *proposers* generates new hypotheses by making incremental changes to the hypotheses found so far. A set of *experts* scores the new hypotheses on criteria of grammatical correctness. These experts include discriminative classifiers for specific error categories, such as articles and prepositions. The decoder model calculates the overall hypothesis score for each hypothesis as a linear combination of the expert scores. The weights of the decoder model are discriminatively trained on a development set of error-annotated sentences. The highest scoring hypotheses are kept in the search beam for the next iteration. This search procedure continues until the beam is empty or the maximum number of iterations has been reached. The highest scoring hypothesis is returned as the sentence-level correction. We evaluate our proposed decoder in the context of the Helping Our Own (HOO) shared task on grammatical error correction (Dale and Kilgarriff, 2011). Our decoder achieves an F_1 score of 25.48% which improves upon the current state of the art.

The remainder of this paper is organized as follows. The next section gives an overview of related work. Section 3 describes the proposed beam-search decoder. Sections 4 and 5 describe the experimental setup and results, respectively. Section 6 provides further discussion. Section 7 concludes the paper.

2 Related Work

In this section, we summarize related work in grammatical error correction. For a more detailed review, the readers can refer to (Leacock et al., 2010).

The classification approach to error correction has mainly focused on correcting article and preposition errors (Knight and Chander, 1994; Han et al., 2006; Chodorow et al., 2007; Tetreault and Chodorow, 2008; Gamon, 2010; Dahlmeier and Ng, 2011b; Rozovskaya and Roth, 2011). The advantage of the classification approach is that it can make use of powerful machine learning algorithms in connection with arbitrary features from the sentence context. Typical features include surrounding N-grams, part-of-speech (POS) tags, chunks, etc. In fact, a considerable amount of research effort has been invested in finding better features.

The SMT approach to error corrections has received comparatively less attention. Brockett *et al.* (2006) use an SMT system to correct errors involving mass noun errors. Because no large annotated learner corpus was available, the training data was created artificially from non-learner text. Lee and Seneff (2006) describe a lattice-based correction system with a domain-specific grammar for spoken utterances from the flight domain. The work in (Désilets and Hermet, 2009) uses simple round-trip translation with a standard SMT system to correct grammatical errors. Dahlmeier and Ng (2011a) correct collocation errors using phrase-based SMT and paraphrases induced from the writer's native language. Park and Levy (2011) propose a noisy channel model for error correction. While their motivation to correct whole sentences is similar to ours, their proposed generative method differs substantially from our discriminative decoder. Park and Levy's model does not allow the use of discriminative expert classifiers as our decoder does, but instead relies on a bigram language model to find grammatical corrections. Indeed, they point out that the language model often fails to distinguish grammatical and ungrammatical sentences.

To the best of our knowledge, our work is the first discriminatively trained decoder for whole-sentence grammatical error correction.

3 Decoder

In this section, we describe the proposed beam-search decoder and its components.

The task of the decoder is to find the best hypothesis (i.e., the best corrected sentence) for a given input sentence. To accomplish this, the decoder needs to be able to perform two tasks: generating new hypotheses from current ones, and discriminating good hypotheses from bad ones. This is achieved by two groups of modules which we call *proposers* and *experts*, respectively. Proposers take a hypothesis and generate a set of new hypotheses, where each new hypothesis is the result of making an incremental change to the current hypothesis. Experts score hypotheses on particular aspects of grammaticality. This can be a general language model score, or the output of classifiers for particular error categories, for example for article and preposition usage. The overall score for a hypothesis is a linear combina-

tion of the expert scores. Note that in our decoder, each hypothesis corresponds to a complete sentence. This makes it easy to apply syntactic processing, like POS tagging, chunking, and dependency parsing, which provides necessary features for the expert models. The highest scoring hypotheses are kept in the search beam for the next iteration. The search ends when the beam is empty or the maximum number of iterations has been reached. The highest scoring hypothesis found during the search is returned as the sentence-level correction. The modular design of the decoder makes it easy to extend the model to new error categories by adding specific proposers and experts without having to change the decoding algorithm.

3.1 Proposers

The proposers generate new hypotheses, given a hypothesis. Because the number of possible hypotheses grows exponentially with the sentence length, enumerating all possible hypotheses is infeasible. Instead, each proposer only makes a small incremental change to the hypothesis in each iteration. A change corresponds to a correction of a single word or phrase. We experiment with the following proposers in this work. Additional proposers for other error categories can easily be added to the decoder.

- **Spelling** Generate a set of new hypotheses, by replacing a misspelled word with each correction proposed by a spellchecker.
- **Articles** For each noun phrase (NP), generate two new hypotheses by changing the observed article. Possible article choices are *a/an*, *the*, and the empty article ϵ.
- **Prepositions** For each prepositional phrase (PP), generate a set of new hypotheses by changing the observed preposition. For each preposition, we define a confusion set of possible corrections.
- **Punctuation insertion** Insert commas, periods, and hyphens based on a set of simple rules.
- **Noun number** For each noun, change its number from singular to plural or vice versa.

3.2 Experts

The experts score hypotheses on particular aspects of grammaticality to help the decoder to discriminate grammatical hypotheses from ungrammatical ones. We employ two types of expert models. The first type of expert model is a standard N-gram language model. The language model expert is not specialized for any particular type of error. The second type of experts is based on linear classifiers and is specialized for particular error categories. We use the following classifier experts in our work. The features for the classifier expert models include features from N-grams, part-of-speech (POS) tags, chunks, web-scale N-gram counts, and dependency parse trees. Additional experts can easily be added to the decoder.

- **Article expert** Predict the correct article for a noun phrase.
- **Preposition expert** Predict the correct preposition for a prepositional phrase.
- **Noun number expert** Predict whether a noun should be in the singular or plural form.

The outputs of the experts are used as hypothesis features in the decoder, as described in the next section.

3.3 Hypothesis Features

Each hypothesis is associated with a vector of real-valued features which are indicators of grammaticality and are computed from the output of the expert models. We call these features *hypothesis features* to distinguish them from the features of the expert classifiers. The simplest hypothesis feature is the log probability of the hypothesis under the N-gram language model expert. To avoid a bias towards shorter hypotheses, we normalize the probability by the length of the hypothesis:

$$score_{lm} = \frac{1}{|\mathbf{h}|} \log Pr(\mathbf{h}), \quad (1)$$

where $\mathbf{h}$ is a hypothesis sentence and $|\mathbf{h}|$ is the hypothesis length in tokens.

For the classifier-based experts, we define two types of features. The first is the *average score* of

the hypothesis under the expert model:

$$score_{avg} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{u}^T f(\mathbf{x_i^h}, y_i^{\mathbf{h}})\right), \quad (2)$$

where $\mathbf{u}$ is the expert classifier weight vector, $\mathbf{x_i^h}$ and $y_i^{\mathbf{h}}$ are the feature vector and the observed class, respectively, for the i-th instance extracted from the hypothesis $\mathbf{h}$ (e.g., the i-th NP in the hypothesis for the article expert), and f is a feature map that computes the expert classifier features. The average score reflects how much the expert model "likes" the hypothesis. The second expert score, which we call *delta score*, is the maximum difference between the highest scoring class and the observed class in any instance from the hypothesis:

$$score_{delta} = \max_{i,y}\left(\mathbf{u}^T f(\mathbf{x_i^h}, y) - \mathbf{u}^T f(\mathbf{x_i^h}, y_i^{\mathbf{h}})\right). \quad (3)$$

Generally speaking, the delta score measures how much the model "disagrees" with the hypothesis.

Finally, each hypothesis has a number of *correction count features* that keep track of how many corrections have been made to the hypothesis so far. For example, there is a feature that counts how often the article correction $\epsilon \rightarrow the$ has been applied. We also add aggregated correction count features for each error category, e.g., how many article corrections have been applied in total. The correction count features allow the decoder to learn a bias against over-correcting sentences and to learn which types of corrections are more likely and which are less likely.

3.4 Decoder Model

The hypothesis features described in the previous subsection are combined to compute the score of a hypothesis according to the following linear model:

$$s = \mathbf{w}^T f_E(\mathbf{h}), \quad (4)$$

where $\mathbf{w}$ is the decoder model weight vector and f_E is a feature map that computes the hypothesis features described above, given a set of experts E. The weight vector $\mathbf{w}$ is tuned on a development set of error-annotated sentences using the PRO ranking optimization algorithm (Hopkins and May, 2011).[1]

[1]We also experimented with the MERT algorithm (Och, 2003) but found that PRO achieved better results.

PRO performs decoder parameter tuning through a pair-wise ranking approach. The algorithm starts by sampling hypothesis pairs from the N-best list of the decoder. The metric score for each hypothesis induces a ranking of the two hypotheses in each pair. The task of finding a weight vector that correctly ranks hypotheses can then be reduced to a simple binary classification task. In this work, we use PRO to optimize the F_1 correction score, which is defined in Section 4.2. PRO requires a sentence-level score for each hypothesis. As F_1 score is not decomposable, we optimize sentence-level F_1 score which serves as an approximation of the corpus-level F_1 score. Similarly, Hopkins and May optimize a sentence-level BLEU approximation (Lin and Och, 2004) instead of the corpus-level BLEU score (Papineni et al., 2002). We observed that optimizing sentence-level F_1 score worked well in practice in our experiments.

3.5 Decoder Search

Given a set of proposers, experts, and a tuned decoder model, the decoder can be used to correct new unseen sentences. This is done by performing a search over possible hypothesis candidates. The decoder starts with the input sentence as the initial hypothesis, i.e., assuming that all words are correct. It then performs a beam search over the space of possible hypotheses to find the best hypothesis correction $\hat{\mathbf{h}}$ for an input sentence $\mathbf{e}$. The search proceeds in iterations until the beam is empty or the maximum number of iterations has been reached. In each iteration, the decoder takes each hypothesis in the beam and generates new hypothesis candidates using all the available proposers. The hypotheses are evaluated by the expert models that compute the hypothesis features and finally scored using the decoder model. As the search space grows exponentially, it is infeasible to perform exhaustive search. Therefore, we prune the search space by only accepting the most promising hypotheses to the pool of hypotheses for future consideration. If a hypothesis has a higher score than the best hypothesis found in previous iterations, it is definitely added to the pool. Otherwise, we use a simulated annealing strategy where hypotheses with a lower score can still be accepted with a certain probability which depends on the difference between the hypothesis score and

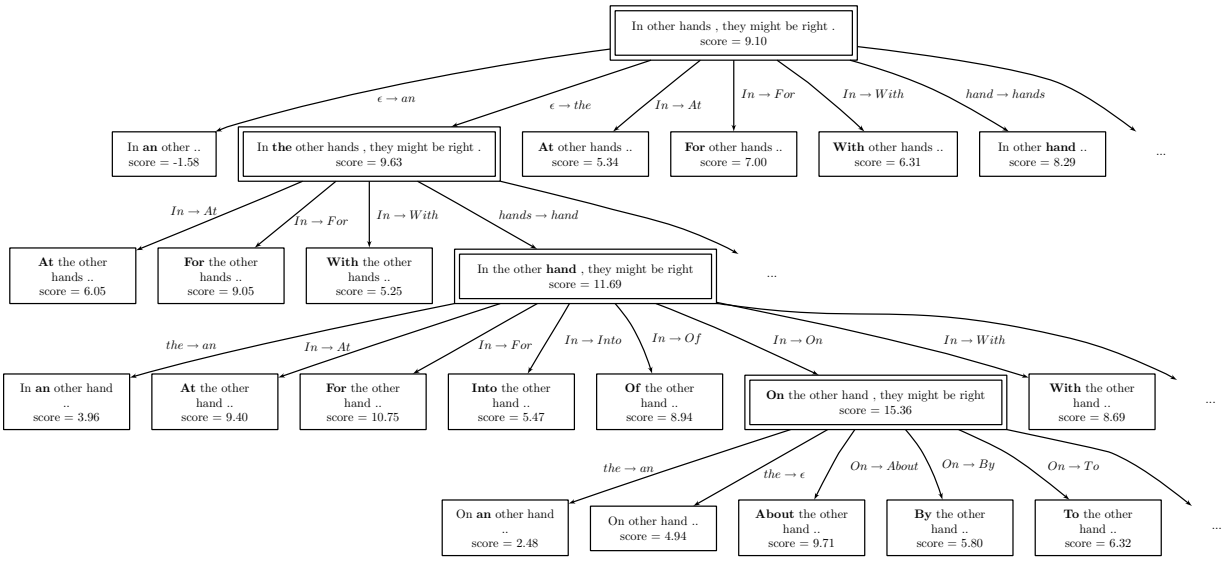

Figure 1: Example of a search tree produced by the beam-search decoder for the input *In other hands, they might be right.* The highest scoring hypothesis found is *On the other hand, they might be right*. Some hypotheses are omitted due to space constraints.

the score of the best hypothesis and the "temperature" of the system. We lower the temperature after each iteration according to an exponential cooling schedule. Hypotheses that have been explored before are not considered again to avoid cycles in the search. From all hypotheses in the pool, we select the top k hypotheses and add them to the beam for the next search iteration. The decoding algorithm is shown in Algorithm 1. The decoder can be considered an *anytime algorithm* (Russell and Norvig, 2010), as it has a current best hypothesis correction available at any point of the search, while gradually improving the result by searching for better hypotheses. An example of a search tree produced by our decoder is shown in Figure 1.

The decoding algorithm shares some similarities with the beam-search algorithm frequently used in SMT. There is however a difference between SMT decoding and grammar correction decoding that is worth pointing out. In SMT decoding, every input word needs to be translated exactly once. In contrast, in grammar correction decoding, the majority of the words typically do *not* need any correction (in the HOO data, for example, there are on average 6 errors per 100 words). On the other hand, some words might require *multiple* corrections, for example spelling correction followed by noun number correction. Errors can also be inter-dependent, where correcting one word makes it necessary to change another word, for example to preserve agreement. Our decoding algorithm has the option to correct some words multiple times, while leaving other words unchanged.

4 Experiments

We evaluate our decoder in the context of the HOO shared task on grammatical error correction. The goal of the task is to automatically correct errors in academic papers from NLP. The readers can refer to the overview paper (Dale and Kilgarriff, 2011) for details. We compare our proposed method with two baselines: a phrase-based SMT system (described in Section 4.3) and a pipeline of classifiers (described in Section 4.4).

4.1 Data

We split the HOO development data into an equal sized training (HOO-TRAIN) and tuning (HOO-TUNE) set. The official HOO test data (HOO-TEST) is used for evaluation. In the HOO shared task, participants were allowed to raise objections regarding the gold-standard annotations (corrections) of the test data after the test data was released. As a result, the gold-standard annotations could be biased in fa-

Algorithm 1 The beam-search decoding algorithm. $\mathbf{e}$: original sentence, $\mathbf{w}$: decoder weight vector, P: set of proposers, E: set of experts, k: beam width, M: maximum number of iterations, T, c: initial temperature and cooling schedule for simulated annealing ($0 < c < 1$).

procedure decode($\mathbf{e}$, $\mathbf{w}$, P, E, k, M)

$beam \leftarrow \{\mathbf{e}\}$
$previous \leftarrow \{\mathbf{e}\}$
$\mathbf{h}_{best} \leftarrow \mathbf{e}$
$s_{best} \leftarrow \mathbf{w}^T f_E(\mathbf{h}_{best})$
$i \leftarrow 0$
while $beam \neq \emptyset \wedge i < M$ **do**
  $pool \leftarrow \{\}$
  for all $\mathbf{h} \in beam$ **do**
    for all $p \in P$ **do**
      for all $\mathbf{h}' \in p.propose(h)$ **do**
        if $\mathbf{h}' \in previous$ **then**
          $continue$
        $previous \leftarrow previous \cup \{\mathbf{h}'\}$
        $s_{\mathbf{h}'} \leftarrow \mathbf{w}^T f_E(\mathbf{h}')$
        if $accept(s_{\mathbf{h}'}, s_{best}, T)$ **then**
          $pool \leftarrow pool \cup \{(\mathbf{h}', s_{\mathbf{h}'})\}$
  $beam \leftarrow \emptyset$
  for all $(\mathbf{h}, s_{\mathbf{h}}) \in nbest(pool, k)$ **do**
    $beam \leftarrow beam \cup \{\mathbf{h}\}$
    if $s_{\mathbf{h}} > s_{best}$ **then**
      $\mathbf{h_{best}} \leftarrow \mathbf{h}$
      $s_{best} \leftarrow s_{\mathbf{h}}$
  $T \leftarrow T \times c$
  $i \leftarrow i + 1$
return $\mathbf{h}_{best}$

procedure accept($s_{\mathbf{h}}$, s_{best}, T)

$\delta \leftarrow s_{\mathbf{h}} - s_{best}$
if $\delta > 0$ **then**
  return $true$
if $exp(\frac{\delta}{T}) > random()$ **then**
  return $true$ **else return** $false$

vor of specific systems participating in the shared task. We obtain both the original and the final official gold-standard annotations and report evaluation results on both annotations.

We use the ACL Anthology[2] as training data for the expert models. We crawl all non-OCR documents from the anthology, except those documents that overlap with the HOO data. Section headers, references, etc. are automatically removed. The Web 1T 5-gram corpus (Brants and Franz, 2006) is used for language modeling and collecting web N-gram counts. Table 1 gives an overview of the data sets.

[2] http://www.aclweb.org/anthology-new/

Data Set	Sentences	Tokens
HOO-TRAIN	467	11,373
HOO-TUNE	472	11,435
HOO-TEST	722	18,790
ACL-ANTHOLOGY	943,965	22,465,690

Table 1: Overview of the data sets.

4.2 Evaluation

We evaluate performance by computing precision, recall, and F_1 correction score without bonus as defined in the official HOO report (Dale and Kilgarriff, 2011)[3]. F_1 correction score is simply the F_1 measure (van Rijsbergen, 1979) between the corrections (called *edits* in HOO) proposed by a system and the gold-standard corrections. Let $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ be a set of test sentences and let $\{\mathbf{g}_1, \ldots, \mathbf{g}_n\}$ be the set of gold-standard edits for the sentences. Let $\{\mathbf{h}_1, \ldots, \mathbf{h}_n\}$ be the set of corrected sentences output by a system. One difficulty in the evaluation is that the set of system edits $\{\mathbf{d}_1, \ldots, \mathbf{d}_n\}$ between the test sentences and the system outputs is ambiguous. For example, assume that the original test sentence is *The data is similar with test set.*, the system output is *The data is similar to the test set.*, and the gold-standard edits are two corrections $with \rightarrow to$, $\epsilon \rightarrow the$ that change *with* to *to* and insert *the* before *test set*. The official HOO scorer however extracts a single system edit $with \rightarrow to\ the$ for this instance. As the extracted system edit is different from the gold-standard edits, the system would be considered wrong, although it proposes the *exact same* corrected sentence as the gold standard edits. This problem has also been recognized by the HOO shared task organizers (see (Dale and Kilgarriff, 2011), Section 5).

Our *MaxMatch* (M^2) scorer (Dahlmeier and Ng, 2012) overcomes this problem through an efficient algorithm that computes the set of system edits which has the maximum overlap with the gold-standard edits. We use the M^2 scorer as the main evaluation metric in our experiments. Additionally, we also report results with the official HOO scorer. Once the set of system edits is extracted, precision, recall, and F_1 measure are computed as follows.

[3] "Without bonus" means that a system does not receive extra credit for not making corrections that are considered optional in the gold standard.

$$P = \frac{\sum_{i=1}^{n} |\mathbf{d}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{d}_i|} \quad (5)$$

$$R = \frac{\sum_{i=1}^{n} |\mathbf{d}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{g}_i|} \quad (6)$$

$$F_1 = 2 \times \frac{P \times R}{P + R} \quad (7)$$

We note that the M^2 scorer and the HOO scorer adhere to the same score definition and only differ in the way the system edits are computed. For statistical significance testing, we use sign-test with bootstrap re-sampling (Koehn, 2004) with 1,000 samples.

4.3 SMT Baseline

We build a baseline error correction system, using the MOSES SMT system (Koehn et al., 2007). Word alignments are created automatically on "good-bad" parallel text from HOO-TRAIN using GIZA++ (Och and Ney, 2003), followed by phrase extraction using the standard heuristic (Koehn et al., 2003). The maximum phrase length is 5. Parameter tuning is done on the HOO-TUNE data with the PRO algorithm (Hopkins and May, 2011) implemented in MOSES. The optimization objective is sentence-level BLEU (Lin and Och, 2004). We note that the objective function is not the same as the final evaluation F_1 score. Also, the training and tuning data are small by SMT standards. The aim for the SMT baseline is not to achieve a state-of-the-art system, but to serve as the simplest possible baseline that uses only off-the-shelf software.

4.4 Pipeline Baseline

The second baseline system is a pipeline of classifier-based and rule-based correction steps. Each step takes sentence segmented plain text as input, corrects one particular error category, and feeds the corrected text into the next step. No search or global inference is applied. The correction steps are:

1. Spelling errors
2. Article errors
3. Preposition errors
4. Punctuation errors
5. Noun number errors

We use the following tools for syntactic processing: OpenNLP[4] for POS tagging, YamCha (Kudo and Matsumoto, 2003) for constituent chunking, and the MALT parser (Nivre et al., 2007) for dependency parsing. For language modeling, we use RandLM (Talbot and Osborne, 2007).

For spelling correction, we use GNU Aspell[5]. Words that contain upper-case characters inside the word or are shorter than four characters are excluded from spell checking. The spelling dictionary is augmented with all words that appear at least 10 times in the ACL-ANTHOLOGY data set.

Article correction is cast as a multi-class classification problem. As the learning algorithm, we choose multi-class confidence-weighted (CW) learning (Crammer et al., 2009) which has been shown to perform well for NLP problems with high dimensional and sparse feature spaces. The possible classes are the articles *a*, *the*, and the empty article ϵ. The article *an* is normalized as *a* and restored later using a rule-based heuristic. We consider all NPs that are not pronouns and do not have a non-article determiner, e.g., *this, that*. The classifier is trained on over 5 million instances from ACL-ANTHOLOGY. We use a combination of features proposed by (Rozovskaya et al., 2011) (which include lexical and POS N-grams, lexical head words, etc.), web-scale N-gram count features from the Web 1T 5-gram corpus following (Bergsma et al., 2009), and dependency head and child features. During testing, a correction is proposed if the predicted article is different from the observed article used by the writer, and the difference between the confidence score for the predicted article and the confidence score for the observed article is larger than a threshold. Threshold parameters are tuned via a grid-search on the HOO-TUNE data. We tune a separate threshold value for each class.

Preposition correction and noun number correction are analogous to article correction. They differ only in terms of the classes and the features. For preposition correction, the classes are 36 frequent English prepositions[6]. The features are surrounding

[4] `http://opennlp.sourceforge.net`

[5] `http://aspell.net`

[6] *about, along, among, around, as, at, beside, besides, between, by, down, during, except, for, from, in, inside, into, of, off, on, onto, outside, over, through, to, toward, towards, under,*

lexical N-grams, web-scale N-gram counts, and dependency features following (Tetreault et al., 2010). The preposition classifier is trained on 1 million training examples from the ACL-ANTHOLOGY. For noun number correction, the classes are *singular* and *plural*. The features are lexical N-grams, web-scale N-gram counts, dependency features, the noun lemma, and a binary countability feature. The noun number classifier is trained on over 5 million examples from ACL-ANTHOLOGY. During testing, the singular or plural word surface form is generated using WordNet (Fellbaum, 1998) and simple heuristics. Punctuation correction is done using a set of simple rules developed on the HOO development data.

At the end of every correction step, all proposed corrections are filtered using a 5-gram language model from the Web 1T 5-gram corpus and only corrections that strictly increase the normalized language model score of the sentence are applied.

4.5 Decoder

We experiment with different decoder configurations with different proposers and expert models. In the simplest configuration, the decoder only has the spelling proposer and the language model expert. We then add the article proposer and expert, the preposition proposer and expert, the punctuation proposer, and finally the noun number proposer and expert. We refer to the final configuration with all proposers and experts as the *full decoder model*. Note that error categories are corrected jointly and not in sequential steps as in the pipeline.

To make the results directly comparable to the pipeline, the decoder uses the same resources as the pipeline. As the expert models, we use a 5-gram language model from the Web 1T 5-gram corpus with the Berkeley LM (Pauls and Klein, 2011)[7] in the decoder and the CW-classifiers described in the last section. The spelling proposer uses the same spellchecker as the pipeline, and the punctuation proposer uses the same rules as the pipeline. The beam width and the maximum number of iterations are set to 10. In earlier experiments, we found that larger values had no effect on the result. The simulated annealing temperature T is initialized to 10 and the exponential cooling schedule c is set to 0.9. The decoder weight vector is initialized as follows. The weight for the language model score and the weights for the classifier expert average scores are initialized to 1.0, and the weights for the classifier expert delta scores are initialized to -1.0. The weights for the correction count features are initialized to zero. For PRO optimization, we use the HOO-TUNE data and the default PRO parameters from (Hopkins and May, 2011): we sample 5,000 hypothesis pairs from the N-best list (N = 100) for every input sentence and keep the top 50 sample pairs with the highest difference in F_1 measure. The weights are optimized using MegaM (Daumé III, 2004) and interpolated with the previous weight vector with an interpolation parameter of 0.1. We normalize feature values to avoid having features on a larger scale dominate features on a smaller scale. We linearly scale all hypothesis features to a unit interval $[0, 1]$. The minimum and maximum values for each feature are estimated from the development data. We use an early stopping criterion that terminates PRO if the objective function on the tuning data drops. To balance the skewed data where samples without errors greatly outnumber samples with errors, we give a higher weight to sample pairs where the decoder proposed a valid correction. We found a weight of 20 to work well, based on initial experiments on the HOO-TUNE data. We keep all these parameters fixed for all experiments.

5 Results

The complete results of our experiments are shown in Table 2. Each row contains the results for one error correction system. Each system is scored on the original and official gold-standard annotations, both with the M^2 scorer and the official HOO scorer. This results in four sets of precision, recall, and F_1 scores for each system. The best published result to date on this data set is the UI Run1 system from the HOO shared task. We include their system as a reference point.

We make the following observations. First, the scores on the official gold-standard annotations are higher compared to the original gold-standard annotations. We note that the gap between the two

underneath, until, up, upon, with, within, without

[7] Berkeley LM is written in Java and was easier to integrate into our Java-based decoder than RandLM.

System	Original gold-standard						Official gold-standard					
	M^2 scorer			HOO scorer			M^2 scorer			HOO scorer		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
UI Run1	40.86	11.21	**17.59**	38.13	10.42	**16.37**	54.61	14.57	**23.00**	50.72	13.34	**21.12**
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
SMT	9.84	7.77	**8.68**	15.25	5.31	**7.87**	23.35	7.38	**11.21**	15.82	5.30	**7.93**
Pipeline	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Spelling	50.00	0.79	1.55	40.00	0.64	1.25	50.00	0.76	1.49	40.00	0.61	1.20
+ Articles	30.86	10.23	15.36	28.04	9.55	14.25	34.42	10.97	16.64	31.78	10.41	15.68
+ Prepositions	27.44	11.90	16.60	24.82	11.15	15.38	30.54	12.77	18.01	27.90	12.04	16.82
+ Punctuation	28.91	14.55	19.36 †	26.57	13.91	**18.25** †	32.88	15.99	21.51	30.63	15.41	**20.50**
+ Noun number	28.77	16.13	**20.67** †	24.68	14.22	18.04 †	32.34	17.50	**22.71**	28.36	15.71	20.22
Decoder	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Spelling	36.84	0.69	1.35	22.22	0.41	0.80	36.84	0.66	1.30	22.22	0.42	0.83
+ Articles	19.84	12.59	15.40	17.99	12.00	14.39	22.45	13.72	17.03 ∗	20.70	13.27	16.16
+ Prepositions	22.62	14.26	17.49 ∗	19.30	12.95	15.50	24.84	15.14	18.81 ∗	21.36	13.78	16.74
+ Punctuation	24.27	18.09	20.73 ∗†	20.40	16.24	18.08	27.13	19.58	22.75 ∗	23.07	17.65	19.99
+ Noun number	30.28	19.17	**23.48** ∗†	24.29	16.24	**19.46** ∗†	33.59	20.53	**25.48** ∗†	27.30	17.55	**21.36** ∗

Table 2: Experimental results on HOO-TEST. Precision, recall, and F_1 score are shown in percent. The best F_1 score for each system is highlighted in bold. Statistically significant improvements ($p < 0.01$) over the pipeline baseline are marked with an asterisk ($*$). Statistically significant improvements over the UI Run1 system are marked with a dagger (†). All improvements of the pipeline and the decoder over the SMT baseline are statistically significant.

annotations is the largest for the UI Run1 system which confirms the suspected bias of the official gold-standard annotations in favor of participating systems. Second, the scores computed with the M^2 scorer are higher than the scores computed with the official HOO scorer. With more error categories and more ambiguity in the edits segmentation, the gap between the scorers widens. In the case of the full pipeline and decoder model, the HOO scorer even shows a *decrease* in F_1 score when the score actually goes up as shown by the M^2 scorer. We therefore focus on the scores of the M^2 scorer from now on. The SMT baseline achieves 8.68% and 11.21% F_1 on the original and official gold standard, respectively. Although the worst system in our experiments, it would still have claimed the third place in the HOO shared task. One problem is certainly the small amount of training data. Another reason is that the phrase-based model is unaware of syntactic structure and cannot express correction rules of the form $NP \rightarrow the\ NP$. Instead, it has to have seen the exact correction rule, e.g., $house \rightarrow the\ house$, in the training data. As a result, the model does not generalize well. The pipeline achieves state-of-the-art results. Each additional correction step improves the score. Our proposed decoder achieves the best result. When only a few error categories are corrected, the pipeline and the decoder are close to each other. When more error categories are added, the gap between the pipeline and the decoder becomes larger. The full decoder model achieves an F_1 score of 23.48% and 25.48% on the original and official gold standard, respectively, which is statistically significantly better than both the pipeline system and the UI Run1 system.

6 Discussion

As pointed out in Section 3.5, the majority of sentences require zero or few corrections. Therefore, the depth of the search tree is typically small. In our experiments, the average depth of the search tree is only 1.9 (i.e., 0.9 corrections per sentence) on the test set. Usually, the search depth will be one larger than the number of corrections made, since the decoder will explore the next level of the search tree before deciding that none of the new hypotheses are better than the current best one. On the other hand, there are many possible hypotheses that can be proposed for any sentence. The breadth of the search tree is therefore quite large. In our experiments, the decoder explored on average 99 hypotheses per sentence on the test set.

PRO iteration	P	R	F_1
1	14.13	20.17	16.62
2	19.71	20.85	20.27
3	23.12	21.03	22.02
4	24.35	20.85	22.47
5	25.53	20.51	22.75
6	26.27	20.34	22.93
7	27.25	20.68	**23.52**
8	26.73	19.83	22.77

Table 3: PRO tuning of the full decoder model on HOO-TUNE

Feature	Weight
$a \rightarrow the$	-1.3660
$a \rightarrow \epsilon$	0.5253
$the \rightarrow a$	-0.9997
$the \rightarrow \epsilon$	0.0532
$\epsilon \rightarrow a$	0.0694
$\epsilon \rightarrow the$	-0.0529

Table 4: Example of PRO-tuned weights for article correction count features for the full decoder model.

We found that PRO tuning is very important to achieve good performance for our decoder. Most importantly, PRO tunes the correction count features that bias the decoder against over-correcting sentences thus improving precision. But PRO is also able to improve recall during tuning. Table 3 shows the trajectory of the performance for the full decoder model during PRO tuning on HOO-TUNE. After PRO tuning has converged, we inspect the learned weight vector and observe some interpretable patterns learned by PRO. First, the language model score and all classifier expert average scores receive positive weights, while all classifier expert delta scores receive negative weights, in line with our initial intuition described in Section 3.3. Second, most correction count features receive negative weights, thus acting as a bias against correction if it is not necessary. Finally, the correction count features reveal which corrections are more likely and which are less likely. For example, article replacement errors are less common in the HOO-TUNE data than article insertions or deletions. The weights learned for the article correction count features shown in Table 4 reflect this.

Although our decoder achieves state-of-the-art results, there remain many error categories which the decoder currently cannot correct. This includes, for example, verb form errors (*Much research (have → has) been put into . . .*) and lexical choice errors (*The (concerned → relevant) relation . . .*). We believe that our decoder provides a promising framework to build grammatical error correction systems that include these types of errors in the future.

7 Conclusion

We have presented a novel beam-search decoder for grammatical error correction. The model performs end-to-end correction of whole sentences with multiple, interacting errors, is discriminatively trained, and incorporates existing classifier-based models for error correction. Our decoder achieves an F_1 correction score of 25.48% on the HOO shared task which outperforms the current state of the art on this data set.

Acknowledgments

This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

References

S. Bergsma, D. Lin, and R. Goebel. 2009. Web-scale N-gram models for lexical disambiguation. In *Proceedings of IJCAI*.

T. Brants and A. Franz. 2006. Web 1T 5-gram corpus version 1.1. Technical report, Google Research.

C. Brockett, W. B. Dolan, and M. Gamon. 2006. Correcting ESL errors using phrasal SMT techniques. In *Proceedings of COLING-ACL*.

M. Chodorow, J. Tetreault, and N.R. Han. 2007. Detection of grammatical errors involving prepositions. In *Proceedings of the 4th ACL-SIGSEM Workshop on Prepositions*.

K. Crammer, M. Dredze, and A. Kulesza. 2009. Multi-class confidence weighted algorithms. In *Proceedings of EMNLP*.

D. Dahlmeier and H.T. Ng. 2011a. Correcting semantic collocation errors with L1-induced paraphrases. In *Proceedings of EMNLP*.

D. Dahlmeier and H.T. Ng. 2011b. Grammatical error correction with alternating structure optimization. In *Proceedings of ACL*.

D. Dahlmeier and H.T. Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of HLT-NAACL*.

R. Dale and A. Kilgarriff. 2011. Helping Our Own: The HOO 2011 pilot shared task. In *Proceedings of the 2011 European Workshop on Natural Language Generation*.

H. Daumé III. 2004. Notes on CG and LM-BFGS optimization of logistic regression. Paper available at `http://pub.hal3.name#daume04cg-bfgs`, implementation available at `http://hal3.name/megam/`.

A. Désilets and M. Hermet. 2009. Using automatic roundtrip translation to repair general errors in second language writing. In *Proceedings of MT-Summit XII*.

C. Fellbaum, editor. 1998. *WordNet: An electronic lexical database*. MIT Press, Cambridge,MA.

M. Gamon. 2010. Using mostly native data to correct errors in learners' writing: A meta-classifier approach. In *Proceedings of HLT-NAACL*.

N.R. Han, M. Chodorow, and C. Leacock. 2006. Detecting errors in English article usage by non-native speakers. *Natural Language Engineering*, 12(02).

M. Hopkins and J. May. 2011. Tuning as ranking. In *Proceedings of EMNLP*.

K. Knight and I. Chander. 1994. Automated postediting of documents. In *Proceedings of AAAI*.

P. Koehn, F.J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of HLT-NAACL*.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL Demonstration Session*.

P. Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of EMNLP*.

T Kudo and Y. Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proceedings of ACL*.

C. Leacock, M. Chodorow, M. Gamon, and J. Tetreault. 2010. *Automated Grammatical Error Detection for Language Learners*. Morgan & Claypool Publishers.

J. Lee and S. Seneff. 2006. Automatic grammar correction for second-language learners. In *Proceedings of Interspeech*.

C.-Y. Lin and F.J. Och. 2004. ORANGE: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of COLING*.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and M. Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13.

F.J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1).

F.J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of ACL*.

Y. A. Park and R. Levy. 2011. Automated whole sentence grammar correction using a noisy channel model. In *Proceedings of ACL*.

A. Pauls and D. Klein. 2011. Faster and smaller N-gram language models. In *Proceedings of ACL-HLT*.

A. Rozovskaya and D. Roth. 2011. Algorithm selection and model adaptation for ESL correction tasks. In *Proceedings of ACL-HLT*.

A. Rozovskaya, M. Sammons, J. Gioja, and D. Roth. 2011. University of Illinois system in HOO text correction shared task. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*.

S. Russell and P. Norvig, 2010. *Artificial Intelligence: A Modern Approach*, chapter 27. Prentice Hall.

D. Talbot and M. Osborne. 2007. Randomised language modelling for statistical machine translation. In *Proceedings of ACL*.

J. Tetreault and M. Chodorow. 2008. The ups and downs of preposition error detection in ESL writing. In *Proceedings of COLING*.

J. Tetreault, J. Foster, and M. Chodorow. 2010. Using parse features for preposition selection and error detection. In *Proceedings of ACL*.

C. J. van Rijsbergen. 1979. *Information Retrieval*. Butterworth, 2nd edition.