Sentence-Level Grammatical Error Identification as Sequence-to-Sequence Correction

Allen Schmaltz **Yoon Kim** **Alexander M. Rush** **Stuart M. Shieber**

Harvard University

{schmaltz@fas,yoonkim@seas,srush@seas,shieber@seas}.harvard.edu

Abstract

We demonstrate that an attention-based encoder-decoder model can be used for sentence-level grammatical error identification for the Automated Evaluation of Scientific Writing (AESW) Shared Task 2016. The attention-based encoder-decoder models can be used for the generation of corrections, in addition to error identification, which is of interest for certain end-user applications. We show that a character-based encoder-decoder model is particularly effective, outperforming other results on the AESW Shared Task on its own, and showing gains over a word-based counterpart. Our final model—a combination of three character-based encoder-decoder models, one word-based encoder-decoder model, and a sentence-level CNN—is the highest performing system on the AESW 2016 binary prediction Shared Task.

1 Introduction

The recent confluence of data availability and strong sequence-to-sequence learning algorithms has the potential to lead to practical tools for writing support. Grammatical error identification is one such application of potential utility as a component of a writing support tool. Much of the recent work in grammatical error identification and correction has made use of hand-tuned rules and features that augment data-driven approaches, or individual classifiers for human-designated subsets of errors. Given a large, annotated dataset of scientific journal articles, we propose a fully data-driven approach for this problem, inspired by recent work in neural machine translation and more generally, sequence-to-sequence learning (Sutskever et al., 2014; Bahdanau et al., 2014; Cho et al., 2014).

The Automated Evaluation of Scientific Writing (AESW) 2016 dataset is a collection of nearly 10,000 scientific journal articles (over 1 million sentences) published between 2006 and 2013 and annotated with corrections by professional, native English-speaking editors. The goal of the associated AESW Shared Task is to identify whether or not a given unedited source sentence was corrected by the editor (that is, whether a given source sentence has one or more grammatical errors, broadly construed).

This system report describes our approach and submission to the AESW 2016 Shared Task, which establishes the current highest-performing public baseline for the binary prediction task. Our primary contribution is to demonstrate the utility of an attention-based encoder-decoder model for the binary prediction task. We also provide evidence of tangible performance gains using a character-aware version of the model, building on the character-aware language modeling work of Kim et al. (2016). In addition to sentence-level classification, the models are capable of intra-sentence error identification and the generation of possible corrections. We also obtain additional gains by using an ensemble of a generative encoder-decoder and a discriminative CNN classifier.

2 Background

Recent work in natural language processing has shown strong results in sequence-to-sequence trans-

formations using recurrent neural network models (Cho et al., 2014; Sutskever et al., 2014). Grammar correction and error identification can be cast as a sequence-to-sequence translation problem, in which an unedited (*source*) sentence is "translated" into a corrected (*target*) sentence in the same language. Using this framework, sentence-level error identification then simply reduces to an equality check between the source and target sentences.

The goal of the AESW shared task is to identify whether a particular sentence needs to be edited (contains a "grammatical" error, broadly construed[1]). The dataset consists of sentences taken from academic articles annotated with corrections by professional editors. Annotations are described via insertions and deletions, which are marked with start and end tags. Tokens to be deleted are surrounded with the deletion start tag <del> and the deletion end tag </del> and tokens to be inserted are surrounded with the insertion start tag <ins> and the insertion end tag </ins>. Replacements (as shown in Figure 1) are represented as deletion-insertion pairs. Unlike the related CoNLL-2014 Shared Task (Ng et al., 2014) data, errors are not labeled with fine-grained types (article or determiner error, verb tense error, etc.).

More formally, we assume a vocabulary $\mathcal{V}$ of natural language word types (some of which have orthographic errors) and a set $\mathcal{Q} = \{\texttt{<ins>}, \texttt{</ins>}, \texttt{~~}, \texttt{~~}\}$ of annotation tags. Given a sentence $\mathbf{s} = [s_1, \ldots, s_I]$, where $s_i \in \mathcal{V}$ is the i-th token of the sentence of length I, we seek to predict whether or not the gold, annotated target sentence $\mathbf{t} = [t_1, \ldots, t_J]$, where $t_j \in \mathcal{Q} \cup \mathcal{V}$ is the j-th token of the annotated sentence of length J, is identical to $\mathbf{s}$. We are given both $\mathbf{s}$ and $\mathbf{t}$ for supervised training. At test time, we are only given access to sequence $\mathbf{s}$. We learn to predict sequence $\mathbf{t}$.

Evaluation of this binary prediction task is via the F_1-score, where the positive class is that indicating an error is present in the sentence (that is, where $\mathbf{s} \neq \mathbf{t}$)[2].

[1] Some insertions and deletions in the shared task data represent stylistic choices, not all of which are necessarily recoverable given the sentence or paragraph context. For the purposes here, we refer to all such edits as "grammatical" errors.

[2] The 2016 Shared Task also included a probabilistic estimation track. We leave for future work the adaptation of our approach to that task.

Evaluation is at the sentence level, but the paragraph-level context for each sentence is also provided. The paragraphs, themselves, are shuffled so that full article context is not available. A coarse academic field category is also provided for each paragraph. Our models described below do not make use of the paragraph context nor the field category, and they treat each sentence independently.

Further information about the task is available in the Shared Task report (Daudaravicius et al., 2016).

3 Related Work

While this is the first year for a shared task focusing on sentence-level binary error identification, previous work and shared tasks have focused on the related tasks of intra-sentence identification and correction of errors. Until recently, standard hand-annotated grammatical error datasets were not available, complicating comparisons and limiting the choice of methods used. Given the lack of a large hand-annotated corpus at the time, Park and Levy (2011) demonstrated the use of the EM algorithm for parameter learning of a noise model using error data without corrections, performing evaluation on a much smaller set of sentences hand-corrected by Amazon Mechanical Turk workers.

More recent work has emerged as a result of a series of shared tasks, starting with the Helping Our Own (HOO) Pilot Shared Task run in 2011, which focused on a diverse set of errors in a small dataset (Dale and Kilgarriff, 2011), and the subsequent HOO 2012 Shared Task, which focused on the automated detection and correction of preposition and determiner errors (Dale et al., 2012). The CoNLL-2013 Shared Task (Ng et al., 2013)[3] focused on the correction of a limited set of five error types in essays by second-language learners of English at the National University of Singapore. The follow-up CoNLL-2014 Shared Task (Ng et al., 2014)[4] focused on the full generation task of correcting all errors in essays by second-language learners.

As with machine translation (MT), evaluation of

[3] http://www.comp.nus.edu.sg/~nlp/conll13st.html

[4] http://www.comp.nus.edu.sg/~nlp/conll14st.html

the full generation task is still an open research area, but a subsequent human evaluation ranked the output from the CoNLL-2014 Shared Task systems (Napoles et al., 2015). The system of Felice et al. (2014) ranked highest, utilizing a combination of a rule-based system and phrase-based MT, with re-ranking via a large web-scale language model. Of the non-MT based approaches, the Illinois-Columbia system was a strong performer, combining several classifiers trained for specific types of errors (Rozovskaya et al., 2014).

4 Models

We use an end-to-end approach that does not have separate components for candidate generation or re-ranking that make use of hand-tuned rules or explicit syntax, nor do we employ separate classifiers for human-differentiated subsets of errors, unlike some previous work for the related task of grammatical error correction.

We next introduce two approaches for the task of sentence-level grammatical error identification: A binary classifier and a sequence-to-sequence model that is trained for correction but can also be used for identification as a side-effect.

4.1 Baseline Convolutional Neural Net

To establish a baseline, we follow past work that has shown strong performance with convolutional neural nets (CNNs) across various domains for sentence-level classification (Kim, 2014; Zhang and Wallace, 2015). We utilize the one-layer CNN architecture of Kim (2014) with the publicly available[5] word vectors trained on the Google News dataset, which contains about 100 billion words (Mikolov et al., 2013). We experiment with keeping the word vectors static (CNN-STATIC) and fine-tuning the vectors (CNN-NONSTATIC). The CNN models only have access to sentence-level labels and are not given correction-level annotations.

4.2 Encoder-Decoder

While it may seem more natural to utilize models trained for binary prediction, such as the aforementioned CNN, or for example, the recurrent network approach of Dai and Le (2015), we hypothesize that training at the lowest granularity of annotations may be useful for the task. We also suspect that the generation of corrections is of sufficient utility for end-users to further justify exploring models that produce corrections in addition to identification. We thus use the Shared Task as a means of assessing the utility of a full generation model for the binary prediction task.

We propose two encoder-decoder architectures for this task. Our word-based architecture (WORD) is similar to that of Luong et al. (2015). Our character-based models (CHAR) still make predictions at the word-level, but use a CNN and a highway network over characters instead of word embeddings as the input to the encoder and decoder, as depicted in Figure 1. We follow past work (Sutskever et al., 2014; Luong et al., 2015) in stacking multiple recurrent neural networks (RNNs), specifically Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) networks, in both the encoder and decoder.

Here, we model the probability of the target given the source, $p(\mathbf{t} \mid \mathbf{s})$, with an *encoder* neural network that summarizes the source sequence and a *decoder* neural network that generates a distribution over the target words and tags at each step given the source.

We start by describing the basic encoder and decoder architectures in terms of the WORD model, and then we describe the CHAR model departures from WORD.

Encoder The encoder reads the source sentence and outputs a sequence of vectors, associated with each word in the sentence, which will be selectively accessed during decoding via a soft attentional mechanism. We use a LSTM network to obtain the hidden states $\mathbf{h}_i^s \in \mathbb{R}^n$ for each time step i,

$$\mathbf{h}_i^s = \text{LSTM}(\mathbf{h}_{i-1}^s, \mathbf{x}_i^s).$$

For the WORD models, $\mathbf{x}_i^s \in \mathbb{R}^m$ is the word embedding for s_i, the i-th word in the source sentence. (The analogue for the CHAR models is discussed below.) The output of the encoder is the sequence of hidden state vectors $[\mathbf{h}_1^s, \ldots, \mathbf{h}_I^s]$. The initial hidden state of the encoder is set to zero (i.e. $\mathbf{h}_0^s \leftarrow \mathbf{0}$).

Decoder The decoder is another LSTM that produces a distribution over the next target word/tag

[5] https://code.google.com/archive/p/word2vec/

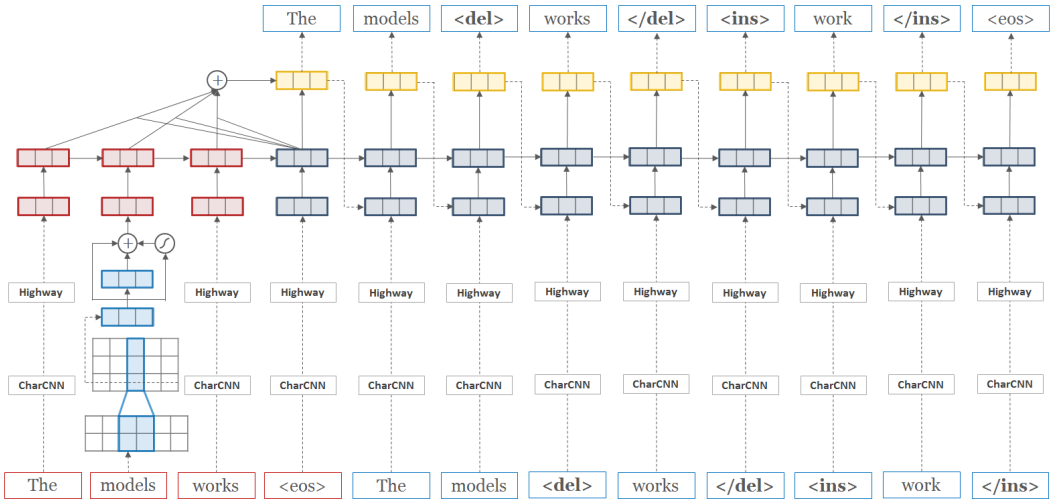

Figure 1: An illustration of the CHAR model architecture translating an example source sentence into the corrected target with a single word replacement. A CNN (here, with three filters of width two) is applied over character embeddings to obtain a fixed dimensional representation of a word, which is given to a highway network (in light blue, above). Output from the highway network is used as input to a LSTM encoder/decoder. At each step of the decoder, its hidden state is interacted with the hidden states of the encoder to produce attention weights (for each word in the encoder), which are used to obtain the context vector via a convex combination. The context vector is combined with the decoder hidden state through a one layer MLP (yellow), after which an affine transformation followed by a softmax is applied to obtain a distribution over the next word/tag. The MLP layer (yellow) is used as additional input (via concatenation) for the next time step. Generation continues until the <eos> symbol is generated.

given the source vectors $[\mathbf{h}^s_1, \ldots, \mathbf{h}^s_I]$ and the previously generated target words/tags $\mathbf{t}_{<j} = [t_1, \ldots t_j]$. Let

$$\mathbf{h}^t_j = \text{LSTM}(\mathbf{h}^t_{j-1}, \mathbf{x}^t_j)$$

be the summary of the target sentence up to the j-th word, where $\mathbf{x}^t_j$ is the word embedding for t_j in the WORD models. The current target hidden state $\mathbf{h}^t_j$ is combined with each of the memory vectors in the source to produce attention weights as follows,

$$u_{j,i} = \mathbf{h}^t_j \cdot \mathbf{W}_\alpha \mathbf{h}^s_i$$
$$\alpha_{j,i} = \frac{\exp u_{j,i}}{\sum_{k \in [1,I]} \exp u_{j,k}}$$

The source vectors are multiplied with the respective attention weights, summed, and interacted with the current decoder hidden state $\mathbf{h}^t_j$ to produce a *context* vector $\mathbf{c}_j$,

$$\mathbf{v}_j = \sum_{i \in [1,I]} \alpha_{j,i} \mathbf{h}^s_i$$
$$\mathbf{c}_j = \tanh(\mathbf{W}[\mathbf{v}_j; \mathbf{h}^t_j])$$

The probability distribution over the next word/tag is given by applying an affine transformation to $\mathbf{c}_j$ followed by a softmax,

$$p(t_{j+1} \,|\, \mathbf{s}, \mathbf{t}_{<j}) = \text{softmax}(\mathbf{U}\mathbf{c}_j + \mathbf{b})$$

Finally, as in Luong et al. (2015), we feed $\mathbf{c}_j$ as additional input to the decoder for the next time step by concatenating it with $\mathbf{x}^t_j$, so the decoder equation is modified to,

$$\mathbf{h}^t_j = \text{LSTM}(\mathbf{h}^t_{j-1}, [\mathbf{x}^t_j; \mathbf{c}_{j-1}])$$

This allows the decoder to have knowledge of previous (soft) alignments at each time step. The decoder hidden state is initialized with the final hidden state of the encoder (i.e. $\mathbf{h}^t_0 \leftarrow \mathbf{h}^s_I$).

Character Convolutional Neural Network For the CHAR models, instead of a word embedding, our input for each word in the source/target sentence is an output from a character-level convolutional neural network (CharCNN) (depicted in Figure 1). Our character model closely follows that of Kim et al. (2016).

Suppose word s_i is composed of characters $[p_1, \ldots, p_l]$. We concatenate the character embeddings to form the matrix $\mathbf{P}_i \in \mathbb{R}^{c \times l}$, where the k-th

column corresponds to the character embedding for p_k (of dimension c).

We then apply a convolution between $\mathbf{P}_i$ and a *filter* $\mathbf{H} \in \mathbb{R}^{c \times w}$ of width w, after which we add a bias and apply a nonlinearity to obtain a *feature map* $\mathbf{f}_i \in \mathbb{R}^{l-w+1}$. The k-th element of $\mathbf{f}_i$ is given by,

$$\mathbf{f}_i[k] = \tanh(\langle \mathbf{P}_i[*, k : k + w - 1], \mathbf{H} \rangle + b)$$

where $\langle \mathbf{A}, \mathbf{B} \rangle = \text{Tr}(\mathbf{A}\mathbf{B}^T)$ is the Frobenius inner product and $\mathbf{P}_i[*, k : k + w - 1]$ is the k-to-$(k + w - 1)$-th column of $\mathbf{P}_i$. Finally, we take the *max-over-time*

$$z_i = \max_k \mathbf{f}_i[k]$$

as the feature corresponding to filter $\mathbf{H}$. We use multiple filters $\mathbf{H}_1, \ldots \mathbf{H}_h$ to obtain a vector $\mathbf{z}_i \in \mathbb{R}^h$ as the representation for a given source/target word or tag. We have separate CharCNNs for the encoder and decoder.

Highway Network Instead of replacing the word embedding $\mathbf{x}_i$ with $\mathbf{z}_i$, we feed $\mathbf{z}_i$ through a *highway network* (Srivastava et al., 2015). Whereas a multilayer perceptron produces a new set of features via the following transformation (given input $\mathbf{z}$),

$$\hat{\mathbf{z}} = f(\mathbf{W}\mathbf{z} + \mathbf{b})$$

a highway network instead computes,

$$\hat{\mathbf{z}} = \mathbf{r} \odot f(\mathbf{W}\mathbf{z} + \mathbf{b}) + (\mathbf{1} - \mathbf{r}) \odot \mathbf{z}$$

where f is a non-linearity (in our models, ReLU), $\odot$ is the element-wise multiplication operator, and $\mathbf{r} = \sigma(\mathbf{W}_r\mathbf{z} + \mathbf{b}_r)$ and $\mathbf{1} - \mathbf{r}$ are called the *transform* and *carry* gates. We feed $\mathbf{z}_i$ into the highway network to obtain $\hat{\mathbf{z}}_i$, which is used to replace the input word embeddings in both the encoder and the decoder.

Inference Exact inference is computationally infeasible for the encoder-decoder models given the combinatorial explosion of possible output sequences, but we follow past work in NMT using beam search. We do not constrain the generation process of words outside insertion tags to words in the source, and each low-frequency holder token generated in the target sentence is replaced with the source token associated with the maximum attention weight. We use a beam size of 10 for all models, with the exception of one of the models in the final system combination, for which we use a beam of size 5, as noted in Section 6.

Note that this model generates corrections, but we are only interested in determining the existence of any error at the sentence-level. As such, after beam decoding, we simply check for whether there were any corrections in the target. However, we found that decoding in this way under-predicts sentence-level errors. It is therefore important to calibrate the weights associated with corrections, which we discuss in Section 5.3.

5 Experiments

5.1 Data

The AESW task data differs from previous grammatical error datasets in terms of scale and genre. To the best of our knowledge, the AESW dataset is the first large-scale, publicly available professionally edited dataset of academic, scientific writing. The training set consists of 466,672 sentences with edits and 722,742 sentences without edits, and the development set contains 57,340 sentences with edits and 90,106 sentences without. The raw training and development datasets are provided as annotated sentences, $\mathbf{t}$, from which the $\mathbf{s}$ sequences may be deterministically derived. There are 143,802 sentences in the Shared Task test set with hidden gold labels, which serve directly as $\mathbf{s}$ sequences.

As part of pre-processing, we treat each sentence independently, discarding paragraph context (which sentences, if any, were present in the same paragraph) and domain information, which is a coarse grouping by the field of the original journal (Engineering, Computer Science, Mathematics, Chemistry, Physics, etc.). We generate Penn Treebank style tokenizations of the input. Case is maintained and digits are not replaced with holder symbols. The vocabulary is restricted to the 50,000 most common tokens, with remaining low frequency tokens replaced with a special <unk> token. The CHAR model can encode but not decode over open vocabularies and hence we do not have any <unk> tokens on the source side of those models. For all of the encoder-decoder models, we replace the low-frequency target symbols during inference as discussed above in Section 4.2.

For development against the provided data with labels, we set aside a 10,000 sentence sample from the original development set for tuning, and use the remaining 137,446 sentences for validation[6]. The encoder-decoder models are given all 466,672 pairs of $\mathbf{s}$ and $\mathbf{t}$ sequences with edits, augmented with varying numbers of pairs without edits. The CHAR+SAMPLE and WORD+SAMPLE models are given a random sample of 200,000 pairs without edits for a total of 666,672 pairs of $\mathbf{s}$ and $\mathbf{t}$ sequences. The CHAR+ALL and WORD+ALL models are given all 722,742 sentences without edits for a total of 1,189,414 pairs of $\mathbf{s}$ and $\mathbf{t}$ sequences. For some of the final testing models, we also train with the development sentences. In these latter cases, all sequence pairs are used. In training all of the encoder-decoder models, as indicated in Section 5.2, we drop sentences exceeding 50 tokens in length.

We also experimented with creating corrected versions of sentences for the CNN. The binary CNN classifiers are given 1,656,086 single-sentence training examples, of which 722,742 are error-free examples (in which $\mathbf{s} = \mathbf{t}$), and the remaining examples are constructed by removing the tags from the annotated sentences, $\mathbf{t}$, to create tag-free examples that contain errors (466,672 instances) and additional error-free examples (466,672 instances).

5.2 Training

Training (along with testing) of all models was conducted on GPUs. Our models were implemented with the Torch[7] framework.

CNN Architecture and training approaches were informed by past work in sentence-level classification using CNNs (Kim, 2014; Zhang and Wallace, 2015). A limited grid search on the development set determined our use of filter windows of width 3, 4, and 5 and 1000 feature maps. We trained for 10 epochs. Training otherwise followed the approach of the correspondingly named CNN-STATIC and CNN-NONSTATIC models of Kim (2014).

encoder-decoder Initial parameter settings (including architecture decisions such as the number of layers and embedding and hidden state sizes) were informed by concurrent work in neural machine translation and existing work such as that of Sutskever et al. (2014) and Luong et al. (2015). We used 4-layer LSTMs with 1000 hidden units in each layer. We trained for 14 epochs with a batch size of 64 and a maximum sequence length of 50. The parameters for the WORD model were uniformly initialized in $[-0.1, 0.1]$, and those of the CHAR model were uniformly initialized in $[-0.05, 0.05]$. The L_2-normalized gradients were constrained to be ≤ 5. Our learning rate schedule started the learning rate at 1 and halved the learning rate after each epoch beyond epoch 10, or once the validation set perplexity no longer improved. The WORD model used 1000-dimensional word embeddings. For CHAR, the character embeddings were 25-dimensional, the filter width was 6, the number of feature maps was 1000, and 2 highway layers were used. The maximum word length was 35 characters for training CHAR. Note that we do not reverse the source ($\mathbf{s}$) sequences, unlike some previous NMT work. Following the work of Zaremba et al. (2014), we employed dropout with a probability of 0.3 between the LSTM layers.

Training both WORD and CHAR on the training set took on the order of a few days using GPUs, with the former being more efficient than the latter. In practice, we used two GPUs for training CHAR due to memory requirements.

5.3 Tuning

Post-hoc tuning was performed on the 10k held-out portion of the development set. In terms of maximizing the F_1-score, this post-hoc tuning was important for these models, without which precision was high and recall was low. We leave to future work alternative approaches to this type of post-hoc tuning.

For the CNN models, after training, we tuned the decision boundary to maximize the F_1-score on the held-out tuning set. Analogously, for the encoder-decoder models, after training the models, we tuned the bias weights (given as input to the final softmax layer generating the words/tags distribution) associated with the four annotation tags via a simple grid search by iteratively running beam search on the tun-

[6] Note that the number of sentences in the final development set without labels posted on CodaLab (http://codalab.org) differed from that originally posted on the AESW 2016 Shared Task website with labels.

[7] http://torch.ch

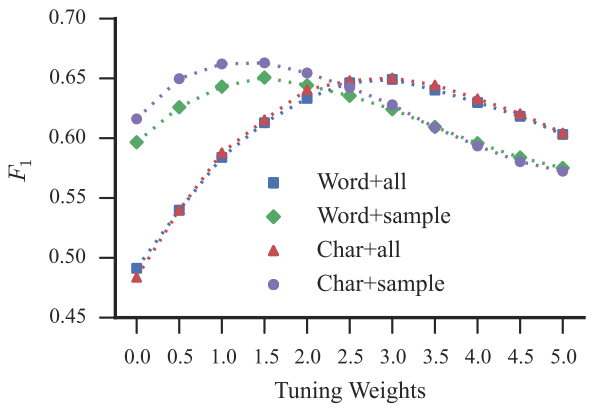

Figure 2: F_1 scores for varying values applied additively to the bias weights of the four annotation tags on the held-out 10k tuning subset.

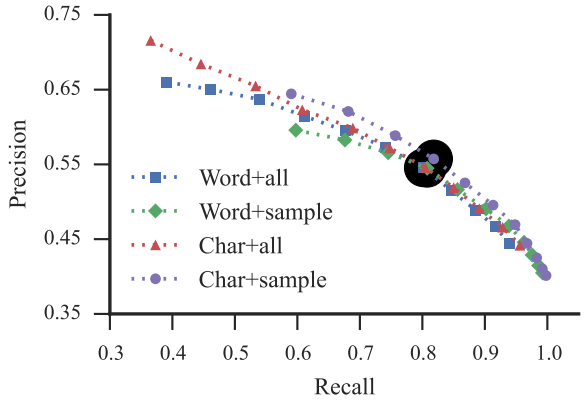

Figure 3: Precision vs. recall trade-off as the bias weights associated with the four annotation tags are varied on the held-out 10k tuning subset. The points yielding maximum F_1 scores are highlighted with black circles.

ing set. Due to the relatively high expense of decoding, we employed a coarse grid search in which the bias weights of the four annotation tags were uniformly varied.

6 Results

Results on the development set, excluding the 10k tuning set, appear in Table 1. Here (and elsewhere) RANDOM is the result of randomly assigning a sentence to one of the binary classes. For the CNN classifiers, fine-tuning the word2vec embeddings improves performance. The encoder-decoder models improve over the CNN classifiers, even though the latter are provided with additional data (via word2vec). The character-based models yield tangible improvements over the word-based models.

For consistency here, we kept the beam size at 10 across models, but subsequent analysis revealed that increasing the beam from 5 to 10 had a negligible effect on overall performance.

Tuning results appear in Figures 2 and 3, illustrating the importance of adjusting the bias weights associated with the annotation tags in balancing precision and recall to maximize the F_1 score. The models trained on all sequence pairs without edits, CHAR+ALL and WORD+ALL, perform particularly poorly without tuning these bias weights, yielding F_1 scores near that of RANDOM before tuning, which corresponds to a weight of 0.0 in Figure 2.

The official development set posted on CodaLab differed slightly from the original development set provided with labels, so we include those results in Table 2 for the encoder-decoder models. Here, evaluation is performed on the CodaLab server, as with the final test submission. The overall relative performance pattern is similar to that of the original development set.

A comparison of our results with other shared task submissions appears in Table 3. (Teams were allowed to submit up to two results.) Our submission, COMBINATION was a simple majority vote at the system level (for each test sentence) of 5 models[8]: (1) a CNN-NONSTATIC model trained with the concatenation of the training and development sets (and using word2vec); (2) a WORD model trained on all sequence pairs in the training and development sets with a beam size of 10 for decoding; (3,4) a CHAR+SAMPLE model trained on the training set, decoding the test set twice, each time with different weight biases (the two highest performing via the grid search over the tuning set) with a beam size of 10; and (5) a CHAR model trained on all sequence pairs in the training and development sets, with training suspended at epoch 9 (out of 14) and a beam size of 5 to meet the Shared Task deadline. For reference, we also include the CodaLab evaluation for just the CHAR+SAMPLE model trained on the training set with a beam size of 10, with the bias

[8]The choice of models was limited to those that were trained and tuned in time for the Shared Task deadline.

Model	Data	Precision	Recall	F_1
RANDOM	N/A	0.3885	0.4992	0.4369
CNN-STATIC	Training+word2vec	0.5349	0.7586	0.6274
CNN-NONSTATIC	Training+word2vec	0.5365	0.7758	0.6343
WORD+ALL	Training	0.5399	0.7882	0.6408
WORD+SAMPLE	Training	0.5394	0.8024	0.6451
CHAR+ALL	Training	0.5400	0.8048	0.6463
CHAR+SAMPLE	Training	0.5526	0.8126	0.6579

Table 1: Experimental results on the development set excluding the held-out 10k tuning subset.

Model	Data	Precision	Recall	F_1
RANDOM	N/A	0.3921	0.5981	0.4736
WORD+ALL	Training	0.5343	0.7577	0.6267
WORD+SAMPLE	Training	0.5335	0.7699	0.6303
CHAR+ALL	Training	0.5351	0.7749	0.6330
CHAR+SAMPLE	Training	0.5469	0.7803	0.6431

Table 2: Results on the official development set. Here, RANDOM was provided by the Shared Task organizers.

Model	Data	Precision	Recall	F_1
RANDOM	N/A	0.3607	0.6004	0.4507
KNOWLET	–	0.6241	0.3685	0.4634
NTNU-YZU	–	0.6717	0.3805	0.4858
HITS	–	0.3765	0.948	0.5389
UW-SU	–	0.4145	0.8201	0.5507
NTNU-YZU	–	0.5025	0.7785	0.6108
CHAR+SAMPLE	Training	0.5112	0.7841	0.6189
COMBINATION	Training+Dev+word2vec	0.5444	0.7413	0.6278

Table 3: Final submitted results on the Shared Task test set. COMBINATION was our final submitted system. RANDOM was provided by the Shared Task organizers. For comparison, we have included the other team submissions from National Taiwan Normal University and Yuan Ze University (NTNU-YZU), the University of Washington and Stanford University (UW-SU), HITS (HITS), and Knowlet (KNOWLET). Teams were allowed to designate up to two final submissions. (The CHAR model trained on the combined training and development set had not finished training by the Shared Task deadline. As such, it was not submitted, but the partially trained model was included in COMBINATION.)

weights being those that generated the highest F_1-score on the 10k tuning set.

7 Discussion

Of particular interest, the CHAR+SAMPLE model performs well, both in terms of performance on the test set relative to other submissions, as well as on the development set relative to the WORD models and the CNN classifiers. It is possible this is due to the ability of the CHAR models to capture some types of orthographic errors.

The empirical results suggest that simply adding additional already correct source-target pairs when training the encoder-decoder models may not boost performance, ceteris paribus, as seen in comparing the performance of CHAR+SAMPLE vs WORD+SAMPLE, and CHAR+ALL vs WORD+ALL. We leave to future work alternative approaches for introducing additional correct (target) sentences, as has been examined for neural machine translation models (Sennrich et al., 2015; Gülçehre et al., 2015).

Our results provide initial evidence to support the hypothesis that training at the lowest granularity of annotation is a more efficient use of data than training against the binary label. In future work, we plan to compare against sentence classification using LSTMs (Dai and Le, 2015) and convolutional models that use correction-level annotations.

Another benefit of the encoder-decoder models is that they can be used to generate corrections (and identify locations of intra-sentence errors) for end-users. However, the added generation capabilities of the encoder-decoder models comes at the expense of considerably longer training and testing times compared to the CNN classifiers.

We found that post-hoc tuning provides a straightforward means of tuning the precision-recall tradeoff for these models, and we speculate (but leave to future work for investigation) that in practice, end-users might prefer greater emphasis placed on precision over recall.

8 Conclusion

We have presented our submission to the AESW 2016 Shared Task, suggesting, in particular, the utility of a neural attention-based model for sentence-level grammatical error identification. Our models do not make use of hand-tuned rules, are not trained with explicit syntactic annotations, and do not make use of individuals classifiers designed for human-designated subsets of errors.

For the encoder-decoder models, modeling at the sub-word level was beneficial, even though predictions were still made at the word level. It would be of interest to push this further to eliminate the need for an initial tokenization step, in order to generalize the approach to other languages, such as Chinese and Japanese.

We plan to examine alternative approaches for training with additional correct (target) sentences. Inducing artificial errors to generate more incorrect (source) sentences is also a direction we intend to pursue.

We leave for future work an analysis of the generation quality of our encoder-decoder models on the AESW dataset and the CoNLL-2014 Shared Task data, as well as user studies to assess whether performance is sufficient in practice to be useful, including the utility of correction vs. identification.

We consider this to be just the beginning of the development of data-driven support tools for writers, and many areas remain to be explored.

Acknowledgments

We would like to thank the organizers of the Shared Task for coordinating the task and making the unique AESW dataset available for research purposes. The Institute for Quantitative Social Science (IQSS) and the Harvard Initiative for Learning and Teaching (HILT) supported earlier, related research that led to our participation in the Shared Task. Jeffrey Ling graciously contributed a torch-based CNN implementation of Kim (2014).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October. Association for Computational Linguistics.

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. *CoRR*, abs/1511.01432.

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*, ENLG '11, pages 242–249, Stroudsburg, PA, USA. Association for Computational Linguistics.

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 Seventh Workshop on Building Educational Applications Using NLP*, pages 54–62, Stroudsburg, PA, USA. Association for Computational Linguistics.

Vidas Daudaravicius, Rafael E. Banchs, Elena Volodina, and Courtney Napoles. 2016. A report on the automatic evaluation of scientific writing shared task. In *Proceedings of the Eleventh Workshop on Innovative Use of NLP for Building Educational Applications*, San Diego, CA, USA, June. Association for Computational Linguistics.

Mariano Felice, Zheng Yuan, Øistein E. Andersen, Helen Yannakoudakis, and Ekaterina Kochmar. 2014. Grammatical error correction using hybrid systems and type filtering. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 15–24, Balti-