Effective Grammatical Error Correction with Neural Machine Translation Techniques

Shubha Guha

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2017

Abstract

This thesis presents the results of applying neural machine translation techniques with a novel objective function adapted to the task of grammatical error correction. It demonstrates that placing more importance on grammatical errors than on grammatically correct text during training succeeds in forcing neural machine translation systems to learn more effectively to correct errors rather than to copy incorrect writing. More specifically, the use of an edit-based weighted token-level cost function dramatically improves recall as hoped, without compromising precision whatsoever.

Acknowledgements

Heartfelt thanks to my supervisor, Dr. Kenneth Heafield, who has never failed to encourage fun. He has been an accessible mentor, on project business and besides, a source of knowledge, experience, and guidance as well as positivity throughout this project. Also thanks to Roman Grundkiewicz who fielded many questions and provided scripts and data.

I thank my father, Angshuman Guha, for his eager mentorship and willing ears at any hour, and my mother, Atreyee Guha, for her tenacious confidence in me.

I owe a deep gratitude to the companions who have carried my spirits throughout this year and who have inspired me to learn and thrive in both dark times and bright.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Shubha Guha*)

Table of Contents

1 Introduction **1**

2 Background **3**

2.1 MaxMatch Scorer 4

2.1.1 Alignment 4

2.1.2 Edit Extraction 5

2.1.3 Scoring Function 6

2.2 GEC Approaches 6

2.2.1 Classifiers 6

2.2.2 Rule-Based 7

2.2.3 Language Models 7

2.3 The Machine Translation Approach 7

2.3.1 Statistical Machine Translation 8

2.3.2 Neural Machine Translation 10

3 Methods **13**

3.1 Model Architecture 13

3.2 Training 14

3.2.1 Objective Function 14

3.2.2 Implementation 15

3.3 Datasets 16

3.3.1 Preprocessing 18

3.3.2 Edit Frequency 19

3.4 Tools 19

4 Results **21**

4.1 Baseline Models 22

4.1.1 A Positively Subterranean Model 23

4.2 Advanced Models 24
4.2.1 Goodbye to False Negatives 25
4.2.2 In Relation to Previous Work 25
4.2.3 Training Times 26
4.2.4 Bonus: Better than Humans? 26

5 Conclusion **29**
5.1 Edit Vector Extraction 29
5.2 Dataset Issues 30

A **31**

Bibliography **33**

Chapter 1

Introduction

Grammatical error correction (GEC) is the task of automatically transforming text with potential grammatical errors into grammatically correct text. Although both input and output text are in the same language, this transformation is similar to the task of translating text from one language to another, so much of recent work has focused on applying the same techniques to GEC that have been successful for translation. While neural machine translation (NMT) techniques have generally outperformed statistical machine translation (SMT) techniques at translation tasks, the same pattern has not been the case for GEC. GEC systems trained using standard NMT methods tend to have low recall, i.e. they learn to copy instead of correcting grammatical errors; consequently, SMT systems remain more effective for GEC. The goal of this project is to improve recall using NMT for GEC.

A key difference that sets apart GEC from MT is that part or all of the input text can be identical to the corresponding output text; not only are both texts in the same language but most input text is error-free in practice (see section 3.3.2 for frequency of grammatical errors in commonly used GEC datasets). An unfortunate consequence is that out-of-the-box NMT techniques result in GEC systems that learn to copy instead of correcting grammatical errors, resulting in poor recall.

Recall, defined as the number of true positives over true positives and false negatives, is a measure intended to capture a high incidence of false negatives. In grammatical error correction, false negatives occur when a system fails to detect and correct grammatical errors and instead simply copies the input text.

Perhaps this issue has been neglected in recent research due to the fact that the standard metric for the past few years is an $F_{0.5}$ score, which places twice the importance on precision as on recall; the idea behind it is that producing output that is

unchanged and equally ungrammatical to the input is preferable to producing output that is changed and more ungrammatical. In this project we focus particularly on recall instead of this $F_{0.5}$ score. We hypothesise that applying a higher cost to grammatical errors during training will prevent the system from learning to copy and will improve its recall.

After modifying the default training objective to relatively punish missed grammatical errors more than unnecessary edits, we successfully forced several models to learn to correct grammatical errors. In general, the greater the relative emphasis, the greater the improvement in recall. As an added bonus, we found that far from sacrificing any precision, our approach ended up improving it as well, proving our strategy an unequivocal success.

The remainder of this document is as follows: Chapter 2 surveys the evolution of recent GEC work, including the development of the task-specific MaxMatch scoring framework. It outlines the methods of phrase-based SMT and NMT and reviews work that has applied these approaches to GEC. Chapter 3 summarises the model architecture used in all of our experiments; training hyperparameter values, the original and modified objective function, convergence criterion; datasets used in this work and commonly in GEC; as well as the tools used in this work. Chapter 4 compares the results of baseline models against models with varying values of the novel training hyperparameter, edit weight. Finally, chapter 5 concludes with further observations and suggestions for future work.

Chapter 2

Background

As described in the research proposal for this project, several GEC shared tasks at the beginning of this decade led to significant advances in research: the Helping Our Own (HOO) shared tasks in 2011 and 2012 (Dale and Kilgarriff, 2011; Dale et al., 2012), and the shared tasks at the 17th and 18th conferences on Computational Natural Language Learning (CoNLL) in 2013 and 2014 (Ng et al., 2013, 2014). In this timeframe, a new task-specific scoring framework was developed (Dahlmeier and Ng, 2012) and the spotlight shifted from highly linguistically motivated and error-specific classifiers and rule-based methods to more generalized machine translation techniques. Error-specific approaches, while they require more linguistic expertise and model complexity to achieve full coverage of grammatical error types, have the advantage of performing particularly well on the specific error types they target. On the other hand, statistical machine translation (SMT) methods have achieved state-of-the-art results on GEC, capturing complex interactions between different error types without requiring the individual construction of as many component models. Some have even attempted to combine the advantages of each technique with hybrid systems (Susanto et al., 2014; Rozovskaya and Roth, 2016).

Meanwhile, machine translation (MT) has experienced its own revolution, from the advent of neural machine translation (NMT) methods around 2013 and 2014 (Kalchbrenner and Blunsom, 2013; Cho et al., 2014b,a; Sutskever et al., 2014) to their adoption by Google and Microsoft last year (Wu et al., 2016). While many GEC researchers have continued to improve the performance of phrase-based SMT (Junczys-Dowmunt and Grundkiewicz, 2016; Chollampatt et al., 2016) and hybrid classifier-SMT systems (Susanto et al., 2014; Rozovskaya and Roth, 2016), those pioneering the application of NMT methods to GEC (Xie et al., 2016; Yuan and Briscoe, 2016) have not been able to

meet the state-of-the-art performance according to the accepted scoring method using the MaxMatch scoring framework, presented in the next section.

2.1 MaxMatch Scorer

Establishing a standard evaluation metric is key to building on prior work and making progress on any research topic. The way that GEC systems have been evaluated in most recent work can be broken down to two components: first, a method to align a pair of sentences and extract the edits necessary to transform the given input to the given output; second, a scoring function based on Levenshtein edit operations (insertion, deletion, and substitution) to assess the accuracy of the transformation, typically an F score between the extracted edits and a set of gold standard edits.

2.1.1 Alignment

There is often more than one way to align two sentences and identify a set of edits that represent their differences. There can be more than one possible set of token-level edits leading to the same resulting sentence, but there can also be phrase-level edits on top of these, resulting in some ambiguity around which system edits to feed into the scoring function.

Dahlmeier and Ng (2012) illustrate the problem with an example from the 2011 HOO shared task:

Source	Our baseline system feeds word into PB-SMT pipeline.
Hypothesis	Our baseline system feeds **a** word into PB-SMT pipeline.

Table 2.1: Example of ambiguous edit extraction from HOO 2011: is the edit an insertion of the token `a`, the substitution of the token `word` with the phrase `a word`, or something else?

The official evaluation script of the shared task extracts the edit $\varepsilon \rightarrow$ `a`, while the gold standard annotation is `word` $\rightarrow$ {`a word`, `words`}. Even though both edits could produce the same output string, there is no overlap between the extracted system edit and the gold standard edit choices, so the hypothesis is considered incorrect.

To solve this problem, Dahlmeier and Ng (2012) designed the MaxMatch (M^2) scoring system to choose from ambiguous system edit possibilities in order to maximize the overlap between system and gold standard edits. The M^2 scorer first constructs an edit lattice by filling in a matrix with the Levenshtein distances between substrings of the tokenised input and output and computing all shortest paths through the matrix that transform the input sentence into the system output. Table 2.2 depicts the Levenshtein matrix between source `i studying informatic .` and hypothesis `i am studying informatics .`

		i	am	studying	informatics	.
	0	1	2	3	4	5
i	1	0	1	2	3	4
studying	2	1	1	1	2	3
informatic	3	2	2	2	2	3
.	4	3	3	3	3	2

Table 2.2: Levenshtein matrix between `i studying informatic .` and `i am studying informatics .`

The shortest paths are represented as a graph with each node corresponding to a cell from the Levenshtein matrix and each edge corresponding to an edit operation: insertion, deletion, substitution, or no change. Since GEC data annotations sometimes include phrase-level like `studying` → `am studying` for the example above, the alignment framework should also allow some phrase-level edits in which some words remain unchanged. However, it should do so within reason, without allowing edits spanning long sequences in which many words remain unchanged, so the number of permitted unchanged words in an edit is limited by a parameter (a reasonable default is 2). These additional edits are added to the edit lattice as new edges that combine adjacent existing edges, as illustrated in 2.1.

2.1.2 Edit Extraction

Based on this edit lattice, the M^2 scorer chooses the complete set of edits from input to output which maximally match the set of gold standard edits. Matches are defined as any edit with the same start and end position in the input sentence and with proposed correction included in the gold standard correction.

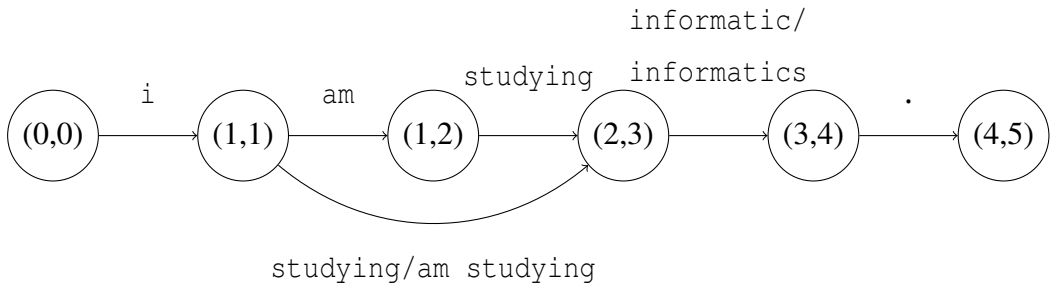

Figure 2.1: Edit lattice from source `i studying informatic .` to hypothesis `i am studying informatics .`

2.1.3 Scoring Function

Initially, GEC researchers continued using F_1 as the scoring function with the results of M^2 alignment and edit extraction, but beginning with CoNLL-2014, it became standard to use the $F_{0.5}$ score (for reasons mentioned in chapter 1), and M^2 score became synonymous with M^2 edit extraction followed by $F_{0.5}$ score.

2.2 GEC Approaches

We can look to the CoNLL shared tasks of 2013 and 2014 to sample the range of possible approaches to GEC. These shared tasks not only popularised the M^2 scoring framework but also introduced datasets that would continue to be used commonly in GEC research to this day.

2.2.1 Classifiers

One of the most popular approaches among submissions to CoNLL-2013 was error-specific machine learning classifiers, one for each of the five grammatical error types included in the task. Five of the submissions were systems based on maximum entropy; others used Naive Bayes, SVMs, perceptrons, and other classifiers. The system with the best F_1 score on the test set (42.14%) consisted of a multi-class averaged perceptron for article or determiner errors, and Naive Bayes for the remaining error types (preposition, noun number, verb form, and subject-verb agreement errors).

Machine learning classifiers remained a common approach in the CoNLL-2014 submission, though almost always in combination with other approaches so as to increase coverage of the 28 error types included in the task. The second best $F_{0.5}$ score on the test set (26.79%), in spite of handling only nine of the 28 error types, was achieved

by a system using "different combinations of averaged perceptron, Naive Bayes, and pattern-based learning trained on different data sets for different error types" (Ng et al., 2014).

2.2.2 Rule-Based

Four CoNLL-2013 submissions were rule-based classifiers and two hybrid systems included rule-based components in a pipeline with other methods. CoNLL-2014 similarly included rule-based approaches, but usually mixed with other approaches. The highest $F_{0.5}$ score achieved in CoNLL-2014 (37.33%) was by a system using rule-based classifiers as a first pass, followed by four steps of ranking hypotheses based on language model probabilities, statistical machine translation, LM ranking again, and type filtering. Hand-crafted rules require extensive linguistic domain expertise Needless to say, purely rule-based techniques no longer appear among the best performing GEC systems.

2.2.3 Language Models

As mentioned for the best performing submission to CoNLL-2014, using language model probabilities to rank hypotheses is another approach, often implemented as a component in a more complex system. It conveniently does not require an annotated grammatical error correction corpus to train an n-gram language model, simply a corpus of grammatically correct text. The result is a way of not only possibly generating new grammatical hypotheses but also comparing their correctness to the source sentence to ensure an improvement.

2.3 The Machine Translation Approach

When CoNLL-2014 increased coverage from five to 28 error types, systems based on error-specific approaches like machine learning and rule-based classifiers became somewhat less practical to build and a larger portion of submissions turned to machine translation techniques, which frame GEC as the translation of ungrammatical text into grammatical text.

In this section, we use the usual MT terminology of translating a foreign/source sentence f into an English/target sentence e, except in GEC both source and target

are in English and the source sentence f may have grammatical errors while the target sentence e has none.

2.3.1 Statistical Machine Translation

The problem of statistical machine translation can be posed using the noisy channel framework and Bayes' Rule as finding the most likely translation $\hat{e}$ for a sentence f in a foreign language:

$$\hat{e} = \underset{e}{\operatorname{argmax}}\ p(e|f) = \underset{e}{\operatorname{argmax}}\ p(e)p(f|e) \tag{2.1}$$

This formulation breaks the system down to two components: a language model $p(e)$ and a translation model (or conditional language model) $p(f|e)$. As the probability of a particular sequence of words occurring in a particular order, the language model represents how correct the target sentence is. Zens et al. (2002) further breaks down the translation model into a lexicon and an alignment model.

In word-based SMT, the translation probability $p(f|e)$ can be represented with an optional sentence length probability, in addition to lexicon probabilities and alignment probabilities:

$$p(f|e) = p(J|e)\sum_{a}\prod_{j=1}^{J}[p(f_j|e_{a_j})p(a_j|a_{j-1},I,J)] \tag{2.2}$$

where J and I are the lengths of the source and target sentences f and e, respectively, and a_j is the index of the target word aligned to source word f_j. The lexicon probability $p(f_j|e_{a_j})$ is the probability that source word f_j translates to target word e_{a_j}. The alignment probability $p(a_j|a_{j-1},I,J)$ is the probability that the position of source word f_j maps to the position of target word e_i, given the alignments of previous words in the source sentence and the lengths of the source and target sentences.

In phrase-based SMT, the segmentation of the sentence pair into K phrases is an additional hidden variable B. Assuming all segmentations have the same probability and allowing only monotone translations:

$$p(f|e) = \prod_{k=1}^{K} p(f_k|e_k) \tag{2.3}$$

where each phrase translation probability $p(f_k|e_k)$ is estimated by its relative frequency:

$$p(f_k|e_k) = \frac{N(f_k,e_k)}{N(e_k)} \tag{2.4}$$

These are counts in the training corpus: $N(e_k)$ is the number of times the phrase e_k occurred, $N(f_k, e_k)$ the number of times f_k appeared as a translation of e_k. Once counted (trained), these estimated probabilities are looked up in a phrase translation table for decoding.

Combining Equations 2.1, 2.3, and 2.4, as well as a translation model scaling factor λ (to indicate relative importance of language model and translation model contributions), the task of monotone translation becomes something more like this:

$$\hat{e} = \underset{e,B}{\operatorname{argmax}} \prod_{i=1}^{I} p(e_i|e_1, e_2, ..., e_{i-1}) \prod_{k=1}^{K} p(f_k|e_k)^{\lambda} \tag{2.5}$$

For non-monotone translations, the task also includes a distance-based phrase reordering (or distortion) probability d (Koehn, 2009):

$$\hat{e} = \underset{e}{\operatorname{argmax}} \prod_{i=1}^{I} p(e_i|e_{<i})^{\lambda_{LM}} \prod_{k=1}^{K} p(f_k|e_k)^{\lambda_\phi} d(start_i - end_{i-1} - 1)^{\lambda_d} \tag{2.6}$$

Each feature has a relative weight λ that must be learned through training.

2.3.1.1 Hybrid Classifier-SMT Systems for GEC

Susanto et al. (2014) surpassed the performance of the best system from CoNLL-2014 using a hybrid system with both error-specific classifiers and SMT. Their best reported $F_{0.5}$ score of 39.39% on the test data from CoNLL-2014 resulted from a combination of four independent GEC systems: two that were pipelines of classifier-based correction steps of six common error types (spelling errors, noun number errors, preposition errors, punctuation errors, article errors, and verb form or subject-verb agreement errors) and two that were SMT models. The pipeline systems differed in the order of the correction steps (swapped noun number and article errors), and the SMT systems differed in phrase table construction (two phrase tables built from two separate corpora versus one phrase table built from the concatenation of the two corpora). Three of the six correction steps used classifiers learned from context features for noun number, prepositions, and articles; two were rule-based classifiers to fix punctuation errors and subject-verb agreement errors; and the first correction step was the output of an open source spell-checker (Jazzy), whose output was filtered using language model probabilities. Neither individual systems nor combinations of one pipeline and one SMT system each could match the performance of the combination of all four systems. The recall of the four-part system was 19.14%, but the two pipeline systems achieved 23.99% and 22.77% individually

More recently, Rozovskaya and Roth (2016) far surpassed even this performance with a classifier-SMT model combination that scored 47.40% on the CoNLL-2014 test set (recall of 25.64%). The classifier used in the final system combination was initially trained on native data, i.e. a corpus of text assumed to be grammatically correct, then "tailored" by artificially introducing grammatical error patterns from a learner corpus, and finally further enhanced by introducing mechanical errors in punctuation, spelling, and capitalization. By running the classifier's output through an SMT system trained on the Lang-8 parallel corpus, they pushed the state-of-the-art to 8 percentage points more than Susanto et al. (2014).

2.3.1.2 SMT Systems for GEC

The state of the art in GEC at the time of this writing was set by Junczys-Dowmunt and Grundkiewicz (2016) with an SMT system tuned to the GEC-specific M^2 metric, i.e. the same $F_{0.5}$ score used to evaluate GEC systems since CoNLL-2014, rather than to the MT-oriented BLEU score as in Susanto et al. (2014) and Rozovskaya and Roth (2016).

Tuning to M^2 with a standard phrase-based SMT system resulted in an impressive score of 46.37% on the CoNLL-2014 test data (recall of 25.05%); when augmented with new GEC-specific features and a large language model trained on the Common Crawl corpus, the system's score increased to 49.49% (recall of 27.98%). The GEC features added onto the standard SMT features include stateless (within phrase) features that captured the edits required to transform an input phrase into a candidate output, stateful (phrase context) features that captured the likeliness of a candidate output phrase in the context of the rest of the output sentence, as well as more fine-grained features capturing edit operations between source and candidate target sentences. Such features have not yet been applied in NMT methods for GEC.

2.3.2 Neural Machine Translation

Current neural machine translation techniques attempt to model $p(e|f)$ directly without the breakdown of language model and translation model. The general idea behind the encoder-decoder architecture introduced by Sutskever et al. (2014), Cho et al. (2014b), and Bahdanau et al. (2014) is to use an RNN to encode the source sentence into a fixed-length vector, then use another RNN to decode this vector into a target sentence. Since LSTMs and GRUs are better at capturing long-range dependencies in sequences,

they are often preferred over vanilla RNNs for tasks such as machine translation.

Bahdanau et al. (2014) proposed an encoder-decoder architecture consisting of a bidirectional LSTM as the encoder and added an attention mechanism to the decoder that behaves somewhat like alignment or distortion in phrase-based SMT to provide a probability distribution over the input positions for any given output position. A context vector calculated at each output position is a weighted average of the encoded representations of each input position, where the weights come from performing a softmax over an alignment model comparing the decoder's hidden state at the previous time step with the encoder's representation at the current time step. This attention mechanism removes the requirement that the input sequence be encoded as a fixed-length vector.

The architecture used in Xie et al. (2016) included a similar attention mechanism in the decoder and a three-layer bidirectional GRU with a pyramid structure as the encoder to reduce the computational complexity resulting from operating at a character level.

2.3.2.1 NMT Systems for GEC

Xie et al. (2016) were the first to apply neural machine translation methods to the GEC task, achieving a score of 40.56% on the CoNLL-2014 test data (recall of 23.77%). They implemented a character-level encoder-decoder recurrent neural network architecture with attention and a language model to prune translation hypotheses for beam search. Their language model was trained on a subset of the Common Crawl corpus, resulting in 2.2 billion n-grams, only a tiny fraction of the more than 500 billion unique n-grams available in the full corpus (Buck et al., 2014). This standard NMT architecture was combined with a mechanism for classifying proposed edits as legitimate or spurious, based on the same understanding mentioned in chapter 1 that it is preferable that a GEC system fail to suggest an edit than that it make a spurious suggestion. Finally, with data augmentation using two common error types according to error distribution statistics for the CoNLL-2014 training set, their best model outperformed Susanto et al. (2014) and set the state of the art at the time.

Yuan and Briscoe (2016) also tried to apply NMT to GEC, outperforming Susanto et al. (2014) but falling short of the performance of the best system by Xie et al. (2016) with a score of 39.90% on the CoNLL-2014 test data (did not report recall). Their implementation operated on the word level instead of the character level as in Xie et al. (2016). However, word-level NMT encounters a problem with handling rare words,

due to the necessity of restricting the size of the known vocabulary so as to efficiently perform each word embedding and compute each softmax at the output layer. This problem is exacerbated in GEC since grammatical errors and spelling mistakes necessarily increase the source vocabulary size and are interpreted as extremely rare words any time a particular error is not systematic and widespread. To handle such "rare words", they aligned unknown target words to their source words using an unsupervised aligner and applied a word-level translation model learned from IBM Model 4 (Brown et al., 1993). However, it seems this approach to handling the rare word problem may not be as effective as character- or subword-level approach as in Xie et al. (2016).

Chapter 3

Methods

We used Nematus (Sennrich et al., 2017) to train attentional encoder-decoder networks with similar architecture to that described in Bahdanau et al. (2014). This chapter lists implementation details including network and training parameters, the novel weighted cost function, and the datasets.

3.1 Model Architecture

Like Bahdanau et al. (2014) and Xie et al. (2016), we use a bidirectional RNN as an encoder. Each input token is represented with the concatenation of the hidden states of the RNN cells in the forward and backward directions; the result is referred to as an annotation for the given input token. The Nematus implementation of the decoder is a modified and simplified version of what Bahdanau et al. (2014) propose: instead of the decoder hidden states being initialised with the last annotation of the encoder's backward RNN, they are initialised with the mean of all annotations; maxout is replaced with tanh in the feedforward hidden layer before the decoder's softmax layer; there are no added biases in both encoder and decoder embedding layers; and the order in which the decoder RNN state updates and generates the next token is switched for a simpler implementation. For reasons of limited time and computational resources, we limited both encoder and decoder depths to a single layer, using the Nematus default of GRU cells (conditional GRU cells in the decoder) instead of LSTM cells as in Bahdanau et al. (2014).

All models used an embedding layer size of 512, hidden layer size of 1000, layer normalisation, and dropout (0.1 for the source and target layers, 0.2 for embedding and hidden layers).

3.2 Training

The following training parameters were left at Nematus default values: maximum sequence length of 100, Adam optimizer, maximum 5000 epochs (which was never reached), maximum 10 million updates (minibatches; also never reached), gradient clipping threshold 1, learning rate 0.0001, maxibatch size 20, cross-entropy objective function (see following subsection for implementation details on modified cross-entropy objective), and early stopping patience of 10, with validation every 10,000 minibatches. Batch size used was 60.

3.2.1 Objective Function

Baseline models were trained to minimize the same cross-entropy loss function used in Xie et al. (2016):

$$L(x,y) = -\sum_{t=1}^{T} \log P(y_t|x,y_{<t}) \tag{3.1}$$

where x is the source sentence and y is the output sentence with T time steps.

3.2.1.1 Edit-Based Weighted Cross-Entropy

As described in the project proposal, in order to improve on recall of error types, we added a weight to the usual cross-entropy loss function that would multiply the loss contribution for time steps when the input and target values were different. In other words, where the learner sentence contained a grammatical error, it was especially important for the system to learn the correct behaviour; where the learner sentence was grammatically correct, it was less important whether the system copied the input or applied an edit.

The weighted cross-entropy loss function used to train more advanced models was the following:

$$L(x,y) = -\sum_{t=1}^{T} \lambda(x_t,y_t) \log P(y_t|x,y_{<t}) \tag{3.2}$$

where the new weight is a function of the input and output at a particular time step t:

$$\lambda(x_t,y_t) = \begin{cases} 1 & \texttt{if } x_t = y_t \\ \Lambda & \texttt{otherwise} \end{cases} \tag{3.3}$$

We report results on the test set for various values of this weight parameter Λ.

3.2.2 Implementation

To identify edits, we used the M^2 scoring script from the CoNLL-2013 shared task to align source and hypothesis sentences, producing a binary edit vector with ones in the positions identified by the aligner as edit words and zeroes in the remaining positions. This edit vector was passed in as an additional input to the Nematus training script, along with an edit weight value specifying the coefficient by which the loss of edit words would be multiplied, i.e. Λ in equation 3.3.

In the following example from the NUCLE dataset (described in section 3.3), the target tokens that are edits are "cause," "is," and "space," so the corresponding edit vector will have ones in those positions (indices 7, 11, and 14 with index origin 0) and zeroes in remaining positions.

Source	this will , if not already , caused problems as there are very limited spaces for us .
Target	this will , if not already , cause problems as there is very limited space for us .

Table 3.1: Example source and target for which edit words are "cause," "is," and "space."

Since cost is calculated over each output token, the edit vector must have the same size as the output sequence, even when the source and target are of different lengths, as in the example below. All in all, of the three possible edit operations (aside from "no edit"), insertions and substitutions result in an output token that we treat as an edit, whereas deletions result in no output token and therefore no output loss contribution that can be multiplied to emphasize training contribution of deleted tokens.

Source	safety is one of the crucial problems that many countries and companies concern .
Target	safety is one of the crucial problems that many countries and companies are concerned about .

Table 3.2: Example source and target for which edit words are "are concerned about."

The training script reads binary edit values as an additional input of the same size as the target values. After computing the usual cross-entropy cost on a given minibatch,

for which edit tokens and non-edit tokens are equally weighted, the script increments this cost by the product of itself with the edit vector of the minibatch and the edit weight previously set as a training parameter (minus one, since the loss from these tokens has already been counted once). As a result, the model is punished more severely for

Algorithm 1 Calculated Edit-Based Weighted Cost

1: **procedure** INCREMENTCOST(cost, edits, edit_weight)
2: weight_matrix $\leftarrow$ edits $*$ (edit_weight $-$ 1)
3: cost $\leftarrow$ cost $+$ (cost $*$ weight_matrix)

incorrectly transforming edit words than non-edit words.

3.3 Datasets

As in Susanto et al. (2014), Chollampatt et al. (2016), and others, we used NUCLE (Dahlmeier et al., 2013) and Lang-8 (Mizumoto et al., 2011; Tajiri et al., 2012) as training data, the CoNLL-2013 test set as development data (Ng et al., 2013), and the CoNLL-2014 test set as test data (Ng et al., 2014).

NUCLE (National University of Singapore Corpus of Learner English) was created expressly for GEC and is publicly available. It contains 1414 essays written by NUS students on a wide variety of topics and corrected by professional English teachers following a standardized annotation schema, illustrated in table 3.3. It includes 27

S	`Therefore , the equipments of biometric identification tend to be in-expensive .`
A	`3 4|||Nn|||equipment|||REQUIRED|||-NONE-|||0`
A	`7 8|||SVA|||tends|||REQUIRED|||-NONE-|||0`
A	`10 11|||Mec|||inexpensive|||REQUIRED|||-NONE-|||0`

Table 3.3: Example learner sentence and annotations from NUCLE corpus.

error categories, though our machine translation approach does not make use of error category information to produce the corrected output.

Lang-8 is a corpus that was extracted from a language learning social network. The 120,000 English texts in the cleaned corpus were written by language learners, usually as diary entries, and corrected by native speakers who were also users on the platform.

As a result of the lack of standardized annotation, this data set is noisier, with target sentences often including parenthetical comments by the annotator instead of strictly correcting the ungrammatical sentence. Two examples are listed in table 3.4.

Source 1	The entertainment was to be at his wedding .
Truth 1	There will be entertainment at this wedding . (sorry , I am not sure what you wanted to say here)
Source 2	They were more expensive compared to others .
Truth 2	They were more expensive compared to others . (this pronoun is unclear " I searched the internet and found the same items at a cheaper price . " is better)

Table 3.4: Training samples from Lang-8 in which target sentences include annotator's commentary in addition to or instead of strict correction of grammatical errors.

We considered the disadvantages of such noisy data and ultimately decided that they were outweighed by the advantage of an aggregate training corpus of 2.2 million sentences: among other benefits, a significantly larger corpus would result in fewer issues related to rare word handling.

Other examples of noise in Lang-8 as well as other datasets used in this work are hyperlinks, citations, and other forms of text for which grammatical error correction seems inapplicable since they are not subject to English grammar rules. Some examples are in table 3.5. These were preprocessed just as any other standard text (preprocessing steps are described next in subsection 3.3.1) and ultimately treated similarly to how rare words would be treated.

Sample 1	References : * Peter G.@@ Peterson .
Sample 2	Harvard International Review .
Sample 3	23@@ .3 (Fall 2001) : 66 * Central Provident Fund , from Wikipedia http : //en.wikipedia.org/wiki/@@ C@@ entr@@ al_@@ Provi@@ dent@@ _@@ Fund * e@@ Go@@ v monitor .

Table 3.5: Three consecutive training "sentences" with no edits, since grammatical rules do not apply to this kind of text.

We used the test data from CoNLL-2013 as a validation set. This dataset has 50 additional essays written and annotated similarly to NUCLE. The only purpose of a validation set was to track training progress and decide early stopping.

For our test data, we will use the test set from CoNLL-2014, which contains another 50 essays written and annotated like those in NUCLE. We report recall and $F_{0.5}$ scores on this dataset for consistent comparison against previous work.

3.3.1 Preprocessing

CoNLL and NUCLE datasets first had to be transformed from an annotation format with source sentence and set of edits (shown in 3.3) into parallel texts that could be processed by a sequence-to-sequence model as in neural machine translation. We used a script from Junczys-Dowmunt and Grundkiewicz (2016) to convert CoNLL and NUCLE notation into parallel corpora. All datasets were received already tokenised in the same way as NUCLE. We used the truecase of the Moses SMT framework (Koehn et al., 2007) to normalise casing, then applied byte pair encoding with the open source subword-nmt package. Finally, we had to replace any pipe symbols | in the datasets with a special symbol `<pipe>` because they carry a special meaning in Nematus.

3.3.1.1 Truecasing

The truecaser in the open source mosesdecoder package was first trained on our concatenated NUCLE and Lang-8 training corpus. Based on the frequencies of words in different cases, the truecaser determines when to lowercase sentence-initial tokens. In the forward direction (preprocessing), the trained truecaser lowercases most sentence-initial tokens, and in the reverse direction (postprocessing, applied to system output before evaluation), detruecasing capitalises all sentence-initial tokens.

3.3.1.2 Byte Pair Encoding

Xie et al. (2016) and Yuan and Briscoe (2016) each had different approaches to dealing with the problem of rare and unknown words in GEC. Instead of using character-level encoding or transforming unknown words as a postprocessing step, we apply byte pair encoding using the open source subword-nmt package (Sennrich et al., 2015). Byte pair encoding is a compression algorithm that can be used to segment words into set number of possible subword units based on how frequently these subwords appear as units in the corpus. A dictionary of fifty thousand subword units was generated from

our training dataset and all datasets were preprocessed by applying this encoding. As a consequence, though "token" and "word" might be used interchangeably with regard to the work in this thesis, tokens are not words per se but often subwords extracted from this process.

3.3.2 Edit Frequency

One factor contributing to the failure of standard NMT techniques to learn to correct grammatical errors, mentioned previously in chapter 1, is that the occurrence of grammatical errors in our corpora is far lower than the occurrence of grammatically correct tokens. After generating edit vectors for the full training corpus including NUCLE and Lang-8, we calculated that only 22.32% of all target tokens were edits. Similarly, Junczys-Dowmunt and Grundkiewicz (2016) report that the CoNLL-2013 test set has an grammatical error rate of 14.97% of all tokens.

As stated in chapter 1, the fact that edit tokens are so much more underrepresented than non-edit tokens is a strong justification to give them more importance during training. Despite addressing this imbalance, there remains an apparent discrepancy between training and validation error rates, which may warrant future efforts to use a validation set more representative of the training set or to clean up some of the noise in Lang-8.

3.4 Tools

We used the following open source packages: Nematus (git version 73037e9), subword-nmt (git version fb526f1 to generate byte pair encoding and version 8873136 for NMT training), mosesdecoder (git version dc42bcb for NMT training).

Chapter 4

Results

Several baseline models were trained with the original Nematus training script to confirm that standard NMT training with cross-entropy loss reproducibly and reliably results in GEC systems with poor recall. To compare against these, we trained seven additional models using the modified training script with different values of the novel edit weight parameter: five were expected to have better recall than the baseline models, one roughly the same, and one worse. Not only did we confirm our hypotheses, we also observed a general trend that greater weight on edit words during training leads to greater recall.

Model	P	R	F_1	$F_{0.5}$
Baseline 1	33.19	14.13	19.82	26.14
Baseline 2	33.76	13.62	19.41	26.05
Baseline 3	34.56	14.43	20.36	27.02
Edit weight 0	8.05	2.48	3.79	5.55
Edit weight 1	34.48	14.37	20.29	26.94
Edit weight 2	37.91	18.69	25.04	31.44
Edit weight 3	39.70	21.17	27.61	33.79
Edit weight 4	39.84	22.94	29.12	34.72
Edit weight 5	39.80	26.24	31.63	36.07
Edit weight 6	40.44	**28.57**	33.48	**37.34**

Table 4.1: Summary of results.

Finally, though it is not uncommon for improvements in recall to coincide with deterioration in precision, this was not the case in our results; moreover, precision of

our advanced models were even noticeably better than that of our baselines. In spite of focusing only on improving recall rather than $F_{0.5}$ measure using the M^2 scorer, we ended up demonstrating that NMT performance even as measured by the GEC standard metric can be enhanced.

4.1 Baseline Models

As shown in table 4.1 above, all three models trained using the baseline training script achieved a dismal performance on the CoNLL-2014 test set, no better than 14.4% recall of grammatical errors. As expected, these models have learned often to copy input sequences without correcting grammatical errors (false negatives), and in examples like Sample 3 in table 4.3 even to make edits that are not necessary (false positives).

Source 1	Mizu@@ shima seaside industrial area is especially well known as one of the largest industrial area in Japan .
Truth 1	Mizu@@ shima 's seaside industrial area is especially well known as one of the largest industrial areas in Japan .
Sample 1	Mizu@@ shima seaside industrial area is especially well known as one of the largest industrial areas in Japan .
Source 2	it is very difficult for me to use " listen to " and " hear " properly .
Truth 2	it is very difficult for me to understand the difference between " listen to " and " hear . "
Sample 2	it is very difficult for me to use " listen to " and " hear " properly .

Table 4.2: Model copies input without correcting grammatical errors (false negatives).

Source 3	I learnt English earlier than learning Japanese , but the latter is more skilled than the former .
Truth 3	I learnt English earlier than learning Japanese , but the latter I 'm more skilled at than the former .
Sample 3	I have learnt English earlier than learning the Japanese , but the latter is more skilled than the former .

Table 4.3: Model attempts to correct text that is grammatically correct (false positives).

4.1.1 A Positively Subterranean Model

As an extra assurance of a correct implementation, we trained models using edit weights of 0 and 1. An edit weight of 1 is expected to result in a system performing exactly as the baseline models because weighting the loss of an edit word one time as much as the loss of a non-edit word is equivalent to ignoring whether a word is an edit or not. An edit weight of 0 is expected to result in even poorer performance by effectively ignoring the loss of all edit words altogether and learning from only non-edit words' loss. These predictions are confirmed, as with edit weight 1 the system achieves scores in the same range as the baselines' and with edit weight 0 the system achieves scores near 0.

Source 1	most of time I spend in the living room .
Truth 1	I spend most of my time in the living room .
Sample 1	most of time I spend the living room .
Source 2	do you have a I phone ? ? ? this is very handy !
Truth 2	do you have an iPhone ? this is very handy !
Sample 2	do you have a I phone ? ? ? this is very handy !
Source 3	in a box of Christmas cake instead of cake .
Truth 3	in a Christmas cake box in place of a cake .
Sample 3	in a box of Christmas cake instead of cake .

Table 4.4: Model with edit weight 0 copies input without correcting grammatical errors (false negatives).

Source	but it is not easy to decre@@ se the well@@ bing budget .
Truth	but it is not easy to decrease the welfare budget .
Sample	but it is not easy to Noy the well@@ bing budget .

Table 4.5: Model with edit weight 0 makes spurious edits, producing a hypothesis even more ungrammatical than if it had copied its input.

As illustrated in tables 4.4 and 4.5, the model trained with edit weight 0 learns to copy input text the vast majority of the time, and occasionally also makes edits that result in outputs that are equally or more ungrammatical than the inputs.

4.2 Advanced Models

After training with edit weights greater than 1, systems achieved dramatically better recall and, somewhat surprisingly, better precision too, though the slope of incremental

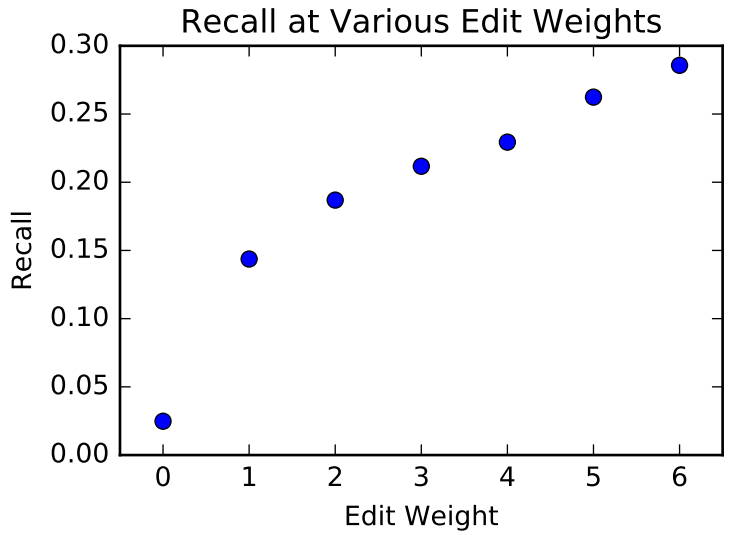

Figure 4.1: The greater the edit weight, the greater the recall.

improvement in precision is notably lower than that in recall.

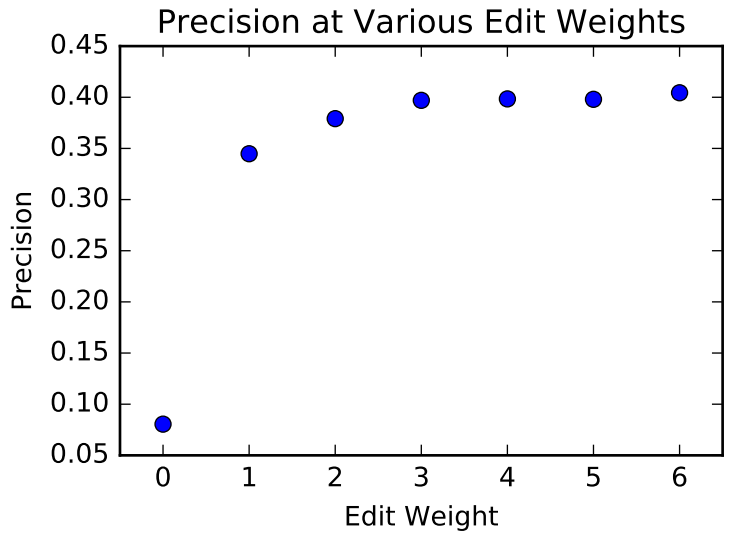

Figure 4.2: The greater the edit weight, the greater the precision.

4.2.1 Goodbye to False Negatives

As recall is intended to capture the relative frequency of false negatives, it is exciting to note that our best system (with recall 28.57% and $F_{0.5}$ score 37.34%) seems to present examples of all but false negatives. The example in table 4.6 includes true negatives such as the phrase "first time" which has no grammatical errors and is correctly copied by the system, true positives such as correcting "to use" to "using," and false positives such as unnecessarily changing "for me , it is the" to "that is my."

Source	for me , it is the first time to use Lang8 .
Truth	for me , it is the first time using Lang8 .
Sample	that is my first time using Lang8 .

Table 4.6: Model with edit weight 6 produces true negatives, true positives, and false positives, but no false negatives (in this example).

Additional examples included in appendix A confirm that generally when the source sentence contains grammatical errors, the system learns to correct at least some of them rather than copying all or most of the time.

4.2.2 In Relation to Previous Work

Comparing the recall of our baseline systems and advanced models is evidence that we have improved on the performance of standard NMT methods for GEC. In addition, our best system has higher recall than any systems surveyed in chapter 2.

	P	R	$F_{0.5}$
Susanto et al. (2014)	53.55	19.14	39.39
Rozovskaya and Roth (2016)	60.17	25.64	47.40
Junczys-Dowmunt and Grundkiewicz (2016)	58.91	25.05	46.37
	61.27	27.98	49.49
Xie et al. (2016)	49.24	23.77	40.56
Yuan and Briscoe (2016)	?	?	39.90
this work	40.44	**28.57**	37.34

Table 4.7: The results of this work in relation to previous work.

On the other hand, while we have shown that precision can improve from applying our novel edit-based weighted cost function, we have not produced a system with precision comparable to the results in table 4.7.

4.2.3 Training Times

There was no particular pattern to the amount of training needed for systems to hit an early stopping patience of 10. Aside from the advanced model with edit weight

Model	Minibatches
Edit weight 0	210k
Edit weight 1	590k
Edit weight 2	710k
Edit weight 3	480k
Edit weight 4	350k
Edit weight 5	400k
Edit weight 6	620k

Table 4.8: No apparent relation between edit weight and training time.

0, which converged after 210k minibatches or parameter updates, training duration ranged from 350k minibatches with edit with 4 to 710k minibatches with edit weight 2. More specifically, increasing edit weight did not appear to speed up training convergence.

4.2.4 Bonus: Better than Humans?

Not only does the best model seem to avoid false negatives more often than the baselines or other advanced models with lower edit weights, weveral interesting examples follow in table 4.9 in which a native English would likely judge the system's output as even more grammatical than the proposed ground truth: in Truth 1, "want" is misspelled; in Truth 2, "our life" is a noun agreement error; in Truth 3, "impossible" is misspelled.

Whether these examples indicate that the system actually outperforms the human annotators of the datasets we used is not a serious question to be decided here, but they do show that finding high quality, uncontroversial data and foolproof evaluation remain open challenges in grammatical error correction work.

Source 1	I think I try to do Twitter for learning English ...
Truth 1	I wan to try to use Twitter to learn English ...
Sample 1	I think I will try to use Twitter to learn English ...
Source 2	they play the important role in our life which can not be substituted .
Truth 2	they play an important role in our life which can not be substituted .
Sample 2	they play an important role in our lives which cannot be substituted .
Source 3	it is almost imp@@ osible for us to keep way from it .
Truth 3	it is almost imp@@ osible for us to keep way from it .
Sample 3	it is almost impossible for us to keep way from it .

Table 4.9: Model with edit weight 6 produces outputs even more grammatical than the target sentences.

Chapter 5

Conclusion

The results presented in this thesis have shown that applying an edit-based weighted cost function in training a neural machine translation framework for grammatical error correction can make learning more effective. Specifically, emphasising edit words more than non-edit words enforces that the system learn to actually correct grammatical errors instead of simply copying text. The result is greater recall and F score with no compromise to precision.

The novel training objective function introduced in this project has allowed us to outperform state-of-the-art grammatical error correction systems on error recall, although our best system remains less advanced on error precision. Future work that uses our edit-based weighted cost function should consider how to combine our recall-oriented loss function with precision-oriented training techniques.

To take our results further, we suggest further exploration of the edit weight hyperparameter space, which we were unable to do in the end due to time and computational resource constraints. We also strongly suggest training multilayer encoder-decoders and optimising other training hyperparameters.

5.1 Edit Vector Extraction

An alternative approach we considered for edit vector extraction uses the network's attention mechanism instead of the M^2 alignment algorithm to extract an edit vector for each target sentence. We decided ultimately to use the M^2 scorer because of its precedent as an established approach for GEC evaluation. Given a reliable way of assessing the alignment accuracy of other approaches such as using the model's attention, it could be an interesting direction in which to explore the use of edit vectors.

5.2 Dataset Issues

In section 3.3, we raised potential issues stemming from the noisiness of Lang-8 and from the low frequency of edit tokens in our training and validation (and probably test) corpora. The latter problem may have been partially mitigated by multiplying the loss of edit tokens, but there also remains the question of whether the error frequency and the general content of our training set (NUCLE and Lang-8 combined) are representative of the validation and test sets used (CoNLL-2013 and CoNLL-2014 test data), and therefore whether the learning of our models is truly generalizable to unseen texts.

After all is said and done, our results and those of most previous work in GEC are only applicable to the writing of second language learners because they are the primary source of GEC datasets. However, GEC in general is relevant to writers of any linguistic background and could benefit from a greater variety of annotated data. With cleaner and more varied data as well as clever training techniques like the modified cost function implemented in this thesis project, this thesis can be error-free; more importantly, GEC researchers can push the envelope on an NLP application that helps us all be better writers.

Appendix A

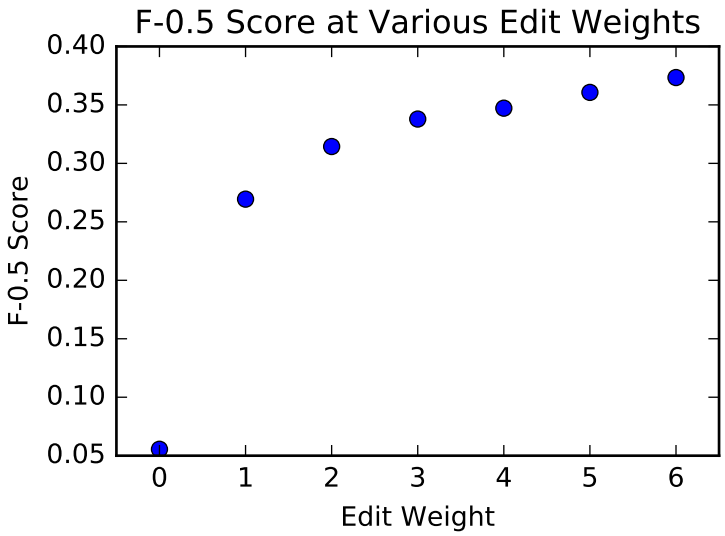

Figure A.1: Generally speaking, $F_{0.5}$ score increases with edit weight, with the best score of 37.34% reached by the system trained with edit weight 6.

Source 1	I felt his ear@@ n@@ ness to study Japanese and his sincerity .
Truth 1	I felt his eagerness to study Japanese and his sincerity .
Sample 1	I felt his eagerness to study Japanese and his sincerity .
Source 2	could you give me a ride to work on Monday ?
Truth 2	could you give me a ride to work on Monday ?
Sample 2	could you give me a ride to work on Monday ?
Source 3	this essay will discuss about whether a carrier of a known genetic risk should tell his or her relatives or not .
Truth 3	this essay will discuss whether a carrier of a known genetic risk should tell his or her relatives or not .
Sample 3	This essay will discuss whether a carrier of a known genetic risk should tell his or her relatives or not .

Table A.1: Samples produced by our best model, trained with edit weight 6.

Bibliography

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural Machine Translation By Jointly Learning To Align and Translate. *Iclr 2015*, pages 1–15.

Brown, P. F., Pietra, S. A. D., Pietra, V. J. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19:263311.

Buck, C., Heafield, K., and Van Ooyen, B. (2014). N -gram Counts and Language Models from the Common Crawl.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the Properties of Neural Machine Translation: EncoderDecoder Approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Chollampatt, S., Taghipour, K., and Ng, H. T. (2016). Neural Network Translation Models for Grammatical Error Correction. pages 2768–2774.

Dahlmeier, D., Ng, H., and Wu, S. (2013). Building a large annotated corpus of learner English: The NUS corpus of learner English. *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 22–31.

Dahlmeier, D. and Ng, H. T. (2012). Better Evaluation for Grammatical Error Correction. *2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572.

Dale, R., Anisimoff, I., and Narroway, G. (2012). HOO 2012: A report on the Preposition and Determiner Error Correction Shared Task. *the Seventh Workshop on Building Educational Applications Using NLP*, pages 54–62.

Dale, R. and Kilgarriff, A. (2011). Helping Our Own: The HOO 2011 Pilot Shared Task. *the Generation Challenges Session at the 13th European Workshop on Natural Language Generation,*, pages 242–249.

Junczys-Dowmunt, M. and Grundkiewicz, R. (2016). Phrase-based Machine Translation is State-of-the-Art for Automatic Grammatical Error Correction. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP-16)*, (1605.06353):1546–1556.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent Continuous Translation Models. pages 1700–1709.

Koehn, P. (2009). *Statistical Machine Translation*. Cambridge University Press.

Koehn, P., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., and Moran, C. (2007). Moses. *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions - ACL '07*, (June):177.

Mizumoto, T., Komachi, M., Nagata, M., and Matsumoto, Y. (2011). Mining Revision Log of Language Learning SNS for Automated Japanese Error Correction of Second Language Learners. pages 147–155.

Ng, H. T., Wu, S. M., Briscoe, T., Hadiwinoto, C., Susanto, R. H., and Bryant, C. (2014). The CoNLL-2014 Shared Task on Grammatical Error Correction. *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, (July):1–14.

Ng, H. T., Wu, Y., and Hadiwinoto, C. (2013). The CoNLL-2013 Shared Task on Grammatical Error Correction. *Computational Natural Language Learning (CoNLL), Shared Task*, pages 1–12.

Rozovskaya, A. and Roth, D. (2016). Grammatical Error Correction: Machine Translation and Classifiers. In *ACL-2016*.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J., and Nadejde, M. (2017). Nematus: a Toolkit for Neural Machine Translation. *In Proceedings of the Demonstrations at the 15th Conference of the European Chapter of the Association for Computational Linguistics, Valencia, Spain.*

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural Machine Translation of Rare Words with Subword Units. pages 1715–1725.

Susanto, R. H., Phandi, P., and Ng, H. T. (2014). System Combination for Grammatical Error Correction. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 951–962.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks.

Tajiri, T., Komachi, M., and Matsumoto, Y. (2012). Tense and aspect error correction for ESL learners using global context. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 2(July):198–202.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, ., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. Technical report.

Xie, Z., Avati, A., Arivazhagan, N., Jurafsky, D., and Ng, A. (2016). Neural Language Correction with Character-Based Attention. *Arxiv*, page 10.

Yuan, Z. and Briscoe, T. (2016). Grammatical error correction using neural machine translation. pages 380–386.

Zens, R., Och, F.-J., and Ney, H. (2002). Phrase-based Statistical Machine Translation. In *Advances in Articial Intelligence. 25th Annual German Conference on Articial Intelligence (KI 02)*, volume 2479, pages 18–32.