The Extensions of Neural Error Correction

Victoria Liao
Christ's College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: pl437@cam.ac.uk

July 26, 2018

Abstract

Grammatical Error Correction (GEC) is a Natural Language Processing (NLP) task that aims to automatically correct grammatical errors in written English sentences (typically by non-native speakers). Recently, it has been proposed that the GEC task should be considered as a *translation* problem, in which a grammatically incorrect source sentence is translated into a correct one (see [Felice et al., 2014, Junczys-Dowmunt and Grundkiewicz, 2014]). The technique has been further developed by the adaptation of Neural Machine Translation (NMT) based on recurrent neural networks (RNNs) (see [Yuan and Briscoe, 2016]).

In 2014, the attention mechanism was introduced in [Bahdanau et al., 2014], and in 2015, this mechanism was proposed to be added to NMT models in [Luong et al., 2015]. The new attention-based NMT model learns to focus on the most relevant information of the input sentence, instead of only using the last hidden state. However, research on attention-based NMT for GEC is not yet well-explored.

Therefore, we implemented an attention-based NMT system to solve the GEC task end-to-end as the baseline of the thesis. We reported the Generalised Language Evaluation Understanding (GLEU) score and Max-Match (M^2) score. We evaluated our models on the test set of First Certificate in English (FCE) (see [Yannakoudakis et al., 2011]), which is extracted from the Cambridge Learner Corpus (CLC). Our baseline system obtains a GLEU score of 66.90 and an $M^2(F_{0.5})$ score of 15.71.

The main contributions of this thesis are four extensions to the baseline, including 1) applying the copy attention mechanism, 2) using Global Vectors for Word Representation (GloVe) word embedding, 3) adding grammatical error detection (GED) preprocessing and 4) adding part-of-speech (PoS) tagging preprocessing. These improvements show a substantial performance gain. We then further explore these individual improvements in combination, which results in a model with the state-of-the-art GLEU score performance on the FCE test dataset. Finally, we report the results of all of our models on the well-known publicly available FCE test set, which can be used for future cross-system comparisons by other researchers.

In more detail, our best model is developed based on the baseline model with the addition of a copy mechanism, PoS tagging preprocessing and GED preprocessing. The GLEU score of the best model is 81.29, and the $M^2(F_{0.5})$ score is 34.37, such that the GLEU score, to the best knowledge of the author, outperforms the state-of-the-art result of [Yannakoudakis et al., 2017] (71.76) on the FCE test set.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

English has long been the most widely accepted and used world language, especially in global trade, international negotiation, world-wide collaboration, information exchange, scientific research and many other areas (see [Van Weijen, 2012]). In 2016, 400 million people spoke English as their first language, and one million people spoke English as their second language [Breene, 2016, Crystal, 2004]). Hence, correct and precise grammar is essential and required for diplomatic documents, ceremonial occasions, and formal conferences.

Grammatical errors take various forms. The most frequently occurring errors are simple and obvious violations of grammatical rules. In general, errors are traditionally identified as occurring on one of five levels: 1) a lexical level, 2) a syntactic level, 3) a semantic level, 4) a discourse structure level, and 5) a pragmatic level [Kukich, 1992]. Three common grammatical errors are described below:

Example 1.0.1 ***Subject-Verb Disagreement***

The subject and verb of a sentence do not agree in terms of person or number.

Incorrect: He ***walk*** *to the college.*

Correct: He ***walks*** *to the college.*

Example 1.0.2 ***Verb Tense Error***

These errors occur when a grammatically incorrect verb tense is used in a sentence.

Incorrect: I ***have seen*** *him yesterday.*

Correct: I ***saw*** *him yesterday.*

Example 1.0.3 ***Noun Disagreement***

Nouns do not agree in number with the terms they are referencing. Multiple objects are mistakenly referred to using a noun in the singular form, or a single object is referred to using a noun in the plural form.

Incorrect: There are a lot of ***restaurant*** *in the college.*

Correct: There are a lot of ***restaurants*** *in the college.*

1.1 Grammatical Error Correction

GEC is the NLP task of correcting grammatical errors in written texts. The task is to build a system that receives a piece of text (typically a sentence) as input, analyses its context to identify and correct any grammatical errors, and finally outputs a corrected version that retains the original meaning. It is particularly helpful to non-native speakers and beginner-level language learners. An example of successful GEC is presented below.

Example 1.1.1 ***NMT for GEC***

Input: ***Its*** *a cold day for October.*

Output: ***It's*** *a cold day for October.*

Ground Truth: ***It's*** *a cold day for October.*

Over the years, information technology (IT) giants have been integrating GEC features in their commercial products. Some of the most well-known GEC examples are Microsoft's Word, Google's Gmail, and Grammarly (a third-party add-on software). The techniques adopted by the above commercial products are rule-based systems (see [Lewis et al., 2004]) that target on limited error types.

1.1.1 Rule-based GEC

Early attempts at GEC employed hand-coded rules
[Gojenola and Oronoz, 2000, Park et al., 1997]. Initial rule-based systems

were based on simple pattern matching and string replacement (see [Heidorn et al., 1982, Bustamante and León, 1996]). Rule-based systems are generally straightforward to implement for certain types of errors and can be very effective and accurate, and hence, most of the current GEC systems employ rule-based mechanisms.

However, rule-based GEC only considers grammatical rules and ignores context, leading to dis-fluency and ambiguity in the translation hypotheses. Additionally, grammatical rules alone can be insufficient to address some complex errors. To overcome the drawbacks of rule-based GEC, and with the advent of large-scale annotated data, several data-driven approaches have been developed, and machine learning algorithms applied to build classifiers in order to correct specific error types [Han et al., 2004, Rozovskaya and Roth, 2011].

1.1.2 Classifier-based GEC

The classifier approach to error correction was prominent for a long time before the advent of machine translation (MT). In the classifier approach, GEC is cast as a multi-class classification problem. To correct a grammatical error, a finite hypothesis set containing all possible correction hypotheses is considered as a set of class labels. Features for the classifier include surrounding n-grams, PoS tags, grammatical relations, and parsing trees. Considering that the most useful features for the classifier depend on the error types, classifiers are trained individually to handle specific error type. The linguistic features are embedded into vectors, and the classifier is trained using various machine learning algorithms.

Some applications of classifier-based GEC have been successful. For example, [Han et al., 2004] trained a maximum entropy classifier to detect article errors and achieved an accuracy of 88%. Moreover, [Tetreault and Chodorow, 2008] used maximum entropy models to correct errors in 34 common English prepositions in the text written by people who speak ESL (with a precision of 0.80 and recall of 0.23).

However, classifier-based GEC has several drawbacks. Firstly, a classifier can only correct a word for one specific error type. Secondly, it ignores the dependencies between the words in a sentence: the classifier implicitly assumes that this context is free of grammatical errors, which is often not the case in practice.

Consequently, MT-based GEC was introduced. This approach considers that in real-world cases, sentences may contain multiple errors of different types and that these errors may depend on each other (see Section 1.1.3).

1.1.3 Machine Translation for GEC

A practical GEC system should be able to correct various types of errors. In more recent research, MT techniques have been used successfully, which automatically translate text from a source language into a target language. GEC can thus be treated as a translation problem from ungrammatical sentences into grammatical sentences. Nowadays, MT-based GEC, including Statistical machine translation (SMT)-based GEC and NMT-based GEC, is the focus of GEC research.

SMT for GEC

SMT was first introduced in [Brockett et al., 2006] for correcting a set of 14 countable/uncountable noun errors made by the speaker of English as a second language (ESL). Subsequently, the SMT-based GEC has become the dominant MT-based GEC approach over the past decade.

The SMT-based GEC model consists of two components: a language model (LM), which assigns a probability $p(y)$ for any target sentence y, and a translation model, which assigns a conditional probability $p(x|y)$, where x is the source [Koehn, 2009]. The LM is learned using a monolingual corpus in the target language. The parameters of the translation model are estimated from a parallel corpus, i.e. the set of source sentences and their corresponding translations into the grammatically correct language.

In the CoNLL-2014 (see [Ng et al., 2014]) shared task on GEC, several top-performing systems employed hybrid approaches. [Felice et al., 2014] proposed a combining rule-based system and a phrase-based SMT system on large LM. The generated hypotheses are ranked using the LM, with the most probable hypothesis being selected as the final corrected version. More recently, [Napoles and Callison-Burch, 2017] proposed a lightweight approach to GEC called Specialised Machine Translation for Error Correction (SMEC), which represents a single model that handles morphological changes, spelling corrections, and phrasal substitutions.

However, compared with an NMT-based GEC system, an SMT-based GEC system has several drawbacks. First, SMT consists of components that are trained separately and then combined during the translation, which might not cooperate well with each other; while the NMT learns a single large neural network that takes a source sentence as input and outputs a translation. As a result, training the NMT systems for end-to-end GEC tasks is much simpler than building SMT-based GEC systems, which require multiple processing steps. Second, an NMT system can learn translation regularities directly from the training data, i.e. we do not need to manually design the features in order to capture them; but in SMT we need to degsign the features. The typical architecture of an NMT-based GEC model is described in the section below.

NMT for GEC

Related Work In 2016, [Yuan and Briscoe, 2016] proposed a neural machine translation approach to GEC using a recurrent neural network to perform sequence-to-sequence (seq2seq) mapping from erroneous to well-formed sentences. In the *translation* task, the sentences with and without grammatical errors are treated as different languages, after which the sentences with grammatical errors are *translated* to sentences without grammatical errors. The technical details of this process are presented in Section 2.2.

Recently, NMT systems have achieved substantial improvements in translation quality [Sutskever et al., 2014, Bahdanau et al., 2014], and attracted growing interest in applying neural systems to GEC. Unlike the phrase-based SMT GEC models mentioned in Section 1.1.3, the seq2seq-based NMT model for GEC can capture long-distance word dependencies, which are crucial for correcting global grammatical errors.

Different network architectures have been proposed for NMT. For example, [Sutskever et al., 2014] used a multilayer LSTM to encode a source sentence into a fixed-sized vector and another LSTM to decode a target sentence from the vector, whereas [Cho et al., 2014] used two Gated Recurrent Unit (GRU) models, one as the encoder and another as the decoder.

Main Challenges and Solutions To achieve better performance on GEC, the NMT model has to address several task-specific challenges. For example, [Yuan and Briscoe, 2016] addressed the massive vocabulary problem by restricting the vocabulary to a limited number of high-frequency words, resort the hypotheses, and refer to standard word translation dictionaries for the translations for out-of-vocabulary (OOV) words. An alternative approach, introduced in [Cho et al., 2014], applies a character-level seq2seq-based NMT model for GEC. However, it cannot effectively leverage word-level information for GEC, even if it is used together with a separate word-based LM.

Most recently, [Ji et al., 2017] proposed a *hybrid neural model* with nested attention layers to solve the OOV word problem. This model consists of 1) a word-based NMT model with an attention mechanism introduced in [Bahdanau et al., 2014]; 2) an additional character-level encoder and decoder; and 3) attention components that focus on OOV words. This nested attention model has proven to be very proficient at correcting global word errors and outperforms previous neural models for GEC as measured on the standard CoNLL-14 benchmark dataset ($M^2(F_{0.5})$ score: 36.04).

Possible Improvements To the best knowledge of the author, the current attention-based NMT systems for GEC can be further improved if we apply data augmentation (adding the information of PoS tags, GED or semantics) and model improvement (using copy mechanism). Also, we can also try the combination of individual improvements. (see Section 4 for details).

1.2 Thesis Objectives

This thesis aims to develop an end-to-end GEC system that is capable of correcting grammatical errors present in the text. As the sentences might contain multiple errors belonging to different error types, we aim to develop robust systems that can automatically detect and correct all types of grammatical errors. We aim to examine and address issues concerning the application of existing NMT techniques to GEC and compare these to others' work. In addition, we aim to explore whether different individual improvements can yield performance gains. Finally, we aim to combine these improvements and report the results.

1.3 Thesis Structure

The remainder of this thesis is structured as follows: chapter 2 introduces the technical background of this thesis, including a conceptual overview of RNNs and their extensions, and the attention-based NMT for GEC. Chapter 2 also introduces the learner corpus FCE dataset and the automatic evaluation metrics used in our thesis.

Chapter 3 describes our approach to the implementation of a baseline NMT-based GEC system, including the system pipeline, hyper-parameter settings, OOV word handling and the loss function. Additionally, we report the performance of the baseline model.

In Chapter 4, we propose some individual improvements that might refine the system. We then report the performance of the new systems, and also investigate combining the improvements for a better GEC model.

Finally, Chapter 5 concludes this thesis and discusses some possible avenues for future research.

Chapter 2

Background

In our GEC project, the NMT for GEC is based on a sequence-to-sequence model that generates a hypothesis sequence when fed a source sequence. This seq2seq for NMT is attention-based and is also based on the RNNs and their extensions since RNNs can handle chronological text sequences. Our NMT model is trained and tested on the FCE dataset, after which the GLEU score and M^2 score are reported.

2.1 RNN Model and its Extensions

2.1.1 Vanilla RNN

RNN is a class of artificial neural networks in which connections between nodes form a directed graph along a sequence. The bidirectional information allows the system to exhibit dynamic temporal behaviour for a time sequence. Unlike feed-forward neural networks, RNNs can use their hidden states (memory) to process sequences of inputs, which makes them suitable for the NMT-based GEC task. For illustrative purposes, a diagram of a one-unit RNN presented in Figure 2.1.

In our GEC task, given a source sentence X, $X = x_1, x_2, ..., x_T$, and a corrected target sentence Y, $Y = y_1, y_2, ..., y_{T'}$, where T and T' are the length of the source and target sentences respectively, we want to map a variable-length input sentence to the variable-length output sentence. Notably, T and T' might be different from each other. In our NMT model, the conditional probability of translating the source sentence X to the target sentence Y is:

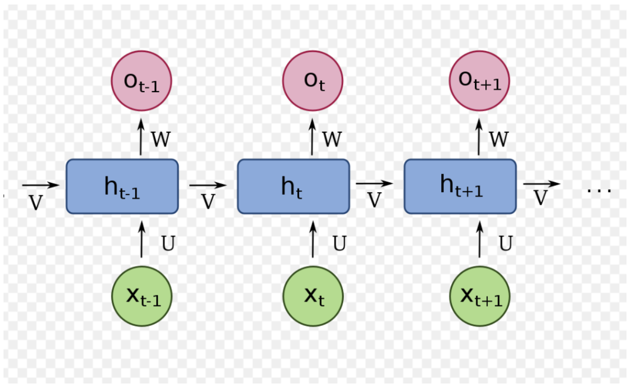


Figure 2.1: A diagram of a one-unit RNN. From the bottom to top: input states, hidden states, output states; and U, V, W are the weights of the network [Deloche, 2017].

$$P(y_1, y_2, ..., y_{T'}|x_1, x_2, ..., x_T). \tag{2.1}$$

The vanilla Elman-style RNN (see [Elman, 1990]) uses a recurrent hidden state to learn sequential information and has an optional output. At each time step t, the recurrent hidden state h_t is calculated based on the current input x_t and the hidden state at the previous time step h_{t-1}, such that:

$$h_t = f(x_t, h_{t-1}), \tag{2.2}$$

where f is a non-linear activation function. Traditionally, a simple *sigmoid* activation function is used to update the recurrent hidden state h_t:

$$h_t = \sigma(Wx_t + U_{t-1}), \tag{2.3}$$

where U and W are the parameter matrices and σ is a smooth, bounded function such as a logistic sigmoid function or a hyperbolic tangent function.

Given the current state h_t, an RNN can be trained to predict a probability distribution over the next word of the sentence:

$$P(x_t|x_1, x_2, ..., x_{t-1}) = g(h_t), \tag{2.4}$$

where g is a non-linear function that outputs the probability of x_t.

Previous work has shown that it is difficult to train an effective RNN to capture long-distance dependencies due to the vanishing or exploding gradient problem (see [Bengio et al., 1994]). When training an RNN model, during back-propagation, we need to compute the gradient of E (the error term) with respect to W_R. The overall error gradient is equal to the sum of the error gradients at each time step. For step t , the RNN model uses the multivariate chain rule to derive the error gradient as:

$$\frac{\delta E_t}{\delta W_R} = \sum_{i=0}^{t} \frac{\delta E_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_t}{\delta h_i}\frac{\delta h_i}{\delta W_R}, \tag{2.5}$$

where $E_t =$ is the error at time t, assuming that E_t is a function of output y_t; $W_R =$ is the recurrent set of weights (other sets of weights are denoted with a different subscript); $h_t =$ is the hidden vector at time t, and the $\frac{\delta h_t}{\delta h_i}$ is calculated as:

$$\frac{\delta h_t}{\delta h_i} = \frac{\delta h_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta h_{t-2}}...\frac{\delta h_{i+1}}{\delta h_i} = \Pi_{k=i}^{t-1}\frac{\delta h_{k+1}}{\delta h_k}. \tag{2.6}$$

If we look at a single one of these terms, i.e., the derivative of h_{k+1} with respect to h_k:

$$\frac{\delta h_{k+1}}{\delta h_k} = diag(f'(W_I x_i + W_R h_{i-1}))W_R, \tag{2.7}$$

where *diag* is the diagonal matrix or the diagonal elements of a matrix.

Therefore, if we want to backpropagate through k time steps, the gradient will be :

$$\frac{\delta h_k}{\delta h_1} = \Pi_i^k diag(f'(W_I x_i + W_R h_{i-1}))W_R. \tag{2.8}$$

If the dominant eigenvalue of the matrix W_R is greater than 1, the gradient explodes; conversely, if it is less than 1, the gradient vanishes
[Pascanu et al., 2013]. Accordingly, previous work has shown that it is difficult to train an effective RNN to capture long-distance dependencies due to the vanishing or exploding gradient problem (see [Bengio et al., 2003]). To solve this vanishing or exploding gradient problem, a long-short-term-memory (LSTM) RNN was introduced (see Section 2.1.2).

2.1.2 LSTM RNN

Although the Elman-style RNN can use information of an arbitrary sequence length, it can only trace back several time steps due to the vanishing gradient, which means that the long-term dependency is lost. However, the long-term dependency is important for capturing context. For example, when the model tries to predict the last word in the text *"I grew up in the* ***UK****... so I speak fluent* ***English****."* it needs the context of *UK*, from earlier in the sentence. Unfortunately, as this gap grows, vanilla RNNs are unable to learn to connect the information.

To solve the vanishing gradient problem in the Elman-style RNN network structure, we employ an LSTM RNN as introduced in [Hochreiter and Schmidhuber, 1997]. The difference between an LSTM model and an Elman-style RNN is is that, in the former, the output of the hidden layers s_t are calculated differently for a specific time t.

In a vanilla RNN, the repeating cells will have a straightforward structure, such as a single tanh layer. The structure of an Elman-style RNN is shown in Figure 2.2.

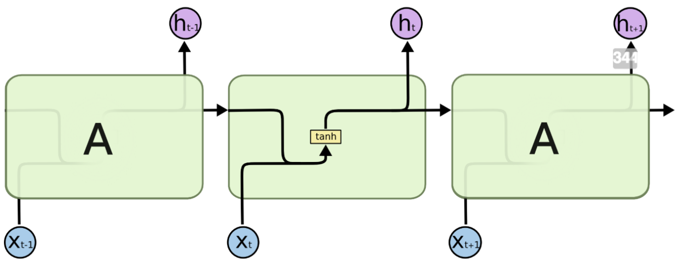


Figure 2.2: The inner structure of an Elman-style RNN, containing a single layer [Collar, 2016], and the repeating mechanism

The mathematical derivation is shown below, and the notations that we used are as follows:

- E_t = Error at time t, assuming that E_t is a function of output y_t.
- W_R = The recurrent set of weights (other sets of weights are denoted with a different subscript).
- $tanh$, σ = The activation function tanh, or sigmoid.
- h_t = The hidden vector at time t.

- U, V = The matrices of hidden states.
- C_t = The LSTM cell state at time t.
- $\tilde{C}_t$ = A vector of the new hypothesis values.
- o_t, f_t, i_t = The LSTM output, forget, and input gates at time t.
- x_t, y_t = The input and output at time t.
- s_t = The output of the hidden states of an LSTM RNN.

A common LSTM unit consists of a *cell*, an *input gate*, an *output gate* and a *forget gate*, the latter of which is not present in a vanilla RNN. The structure of an LSTM RNN is shown in Figure 2.3.

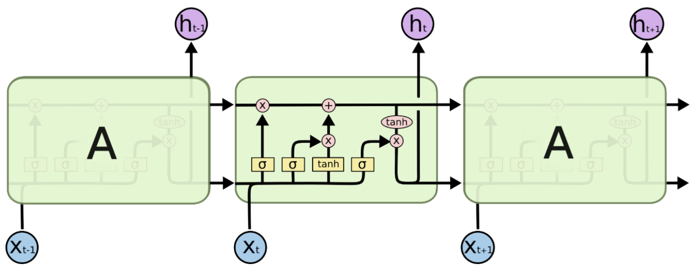


Figure 2.3: The repeating module in an LSTM, which contains four interacting layers [Collar, 2016]

The first step in the LSTM model is to calculate what information should be forgotten from the cell state. This is done by a sigmoid layer called the forget-gate layer. This layer reads h_{t-1} and x_t, and outputs a probability between 0.0 and 1.0 for each number in the cell state C_{t-1}. A 1.0 represents not forgetting the information, while a 0.0 represents completely forgetting the information (see Figure 2.4 and Equation 2.9):

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f). \tag{2.9}$$

The next step is to calculate what new input information should be stored in the cell state. Firstly, a sigmoid layer called the input gate layer calculates which values to update. Next, a tanh layer creates a vector of the new hypothesis values, $\tilde{C}_t$, which can be added to the state (see Figure 2.5 and Equation 2.10 and 2.11):

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i). \tag{2.10}$$

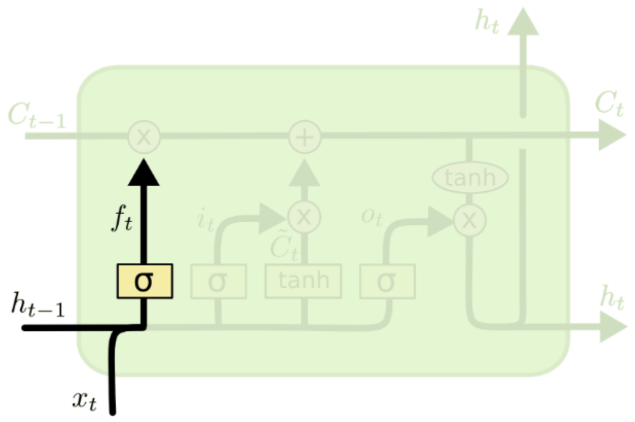


Figure 2.4: LSTM Step 1: Forget gate. The first step in our LSTM RNN is to calculate what information should be forgotten from the cell state. [Collar, 2016]

$$\tilde{C_t} = \tanh(W_c * [h_{t-1}, x_t] + b_C). \tag{2.11}$$

When an LSTM RNN updates the old cell state, C_{t-1}, into the new cell state C_t, it multiplies the old state by f_t (forgetting), then adds $i_t * \tilde{C_t}$. The new hypothesis values are scaled by how much the model calculated to update each state value (see Figure 2.6 and Equation 2.12):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t}. \tag{2.12}$$

Finally, an LSTM module needs to calculate the output, which is based on its cell state but will be filtered. First, it runs a sigmoid layer that calculates what parts of the cell state it is going to output. It then puts the cell state through tanh and multiplies it by the output of the sigmoid gate (see Figure 2.7 and Equations 2.13 and 2.14).

$$o_t = \sigma(W_o[h_{t_1}, x_t] + b_o). \tag{2.13}$$

$$h_t = o_t * \tanh(C_t). \tag{2.14}$$

In the LSTM model, C_t is a function of f_t (the forget gate), i_t (the input

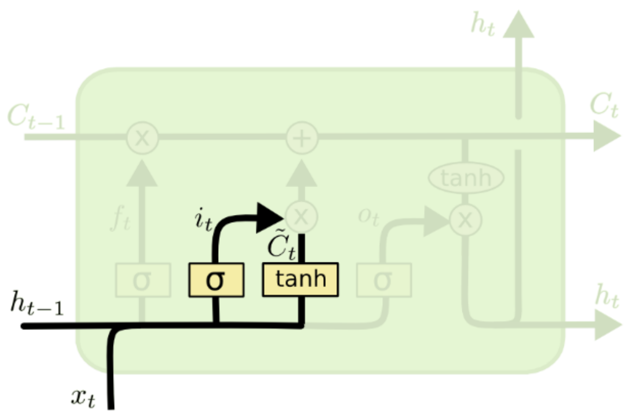


Figure 2.5: LSTM Step 2: Input. The next step is to calculate what new information will be stored in the cell state [Collar, 2016]

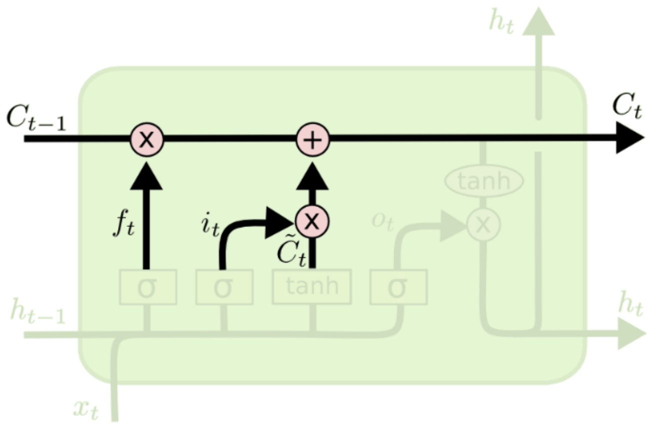


Figure 2.6: LSTM Step 3: Update. When an LSTM RNN updates the old cell state, C_{t-1}, into the new cell state C_t, it multiplies the old state by f_t (forgetting), then adds $i_t * \tilde{C}_t$ [Collar, 2016]

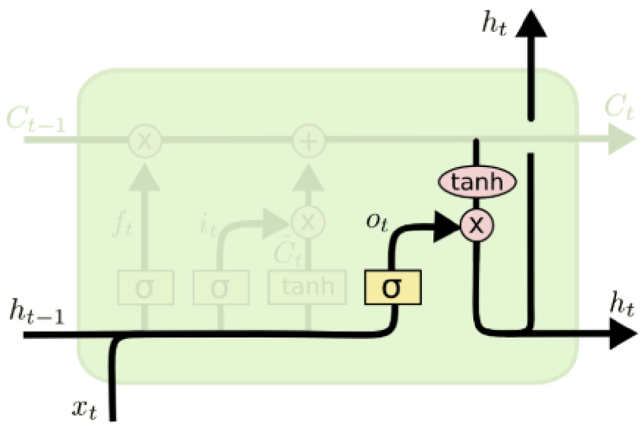


Figure 2.7: LSTM Step 4: Output. An LSTM module needs to calculate the output, which is based on its cell state but will be filtered. [Collar, 2016]

gate), and $\tilde{C}_t$ (the hypothesis cell state), each of which are a function of C_{t-1} [Collar, 2017]. According to the multivariate chain rule:

$$\frac{\delta C_t}{\delta C_{t-1}} = \frac{\delta C_t}{\delta f_t}\frac{\delta f_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta C_{t-1}} + \frac{\delta C_t}{\delta f_t}\frac{\delta i_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta C_{t-1}} + \frac{\delta C_t}{\delta \tilde{C}_t}\frac{\delta \tilde{C}_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta C_{t-1}} + \frac{\delta C_t}{\delta C_{t-1}}. \tag{2.15}$$

The derivative of Equation 2.15 is:

$$\begin{aligned}\frac{\delta C_t}{\delta C_{t-1}} &= C_{t-1}\sigma'(.)W_f * o_{t-1}\tanh'(C_{t-1}) + \tilde{C}_t\sigma'(.)W_i * o_{t-1}\tanh'(C_{t-1})\\ &+ i_t\tanh'(.)W_c * o_{t-1}\tanh'(C_{t-1}) + f_t.\end{aligned} \tag{2.16}$$

Thus, if we start to converge to zero, we can always set the values of f_t higher so as to bring the value of $\frac{\delta C_t}{\delta C_{t-1}}$ closer to 1, thus preventing the gradients from vanishing. Consequently, the f_t is learnt by the LSTM.

2.1.3 Bidirectional LSTM RNN

In this GEC project, bidirectional LSTM (BiLSTM) RNNs were used. The bidirectional mechanism allows the RNN to use both future input and previous input in the prediction; the LSTM mechanism allows the RNN to use both words close to and far away from the current states in its prediction. These features are especially useful because both previous words and following words are essential in GEC, and sometimes significant words are multiple time steps away from the state that requires the prediction.

For example, when a model tries to predict the part-of-speech of *address*, it needs information from both the prior and following words, some of which may be quite far away. In the context *The address is Flat B, A Road, London, UK*, the word *address* is a noun, while in the context of *I will address a speech.* the word *address* is a verb. In both cases, both the past and following words and the LSTM mechanism are necessary if a correct prediction is to be made.

Unlike vanilla RNNs (which are restricted in their ability to use future input information, as the future input cannot be reached from the current state), Bidirectional LSTM (BiLSTM) RNNs use both past and future input, and an LSTM mechanism is applied.

As shown in Figure 2.8, the BiLSTM RNN splits the neurons of a regular LSTM RNN in two directions: one forwards and the other backwards. The outputs of these two states are not connected to the inputs of the opposite states. Then, the BiLSTM RNN connects two hidden layers of opposite directions to the same output.

The forward and backward hidden states at time t in a BiLSTM RNN are:

$$\overrightarrow{h_t} = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h_{t-1}} + \overrightarrow{b}).\overleftarrow{h_t} = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h_{t-1}} + \overleftarrow{b}). \tag{2.17}$$

The output of a BiLSTM is:

$$\hat{y_t} = g(Uh_t + c = g(U[\overrightarrow{h_t}; \overleftarrow{h_t}] + c), \tag{2.18}$$

where h_t is the relationship by which the hidden layer output features are computed at each time-step t, x_t is the input word vector at time t, W is the weight matrix, b is the bias, g is the gradient and $[\overrightarrow{h_t}; \overleftarrow{h_t}]$ is the past and future around a single token.

BiLSTM RNNs are especially useful when much of the context of the past

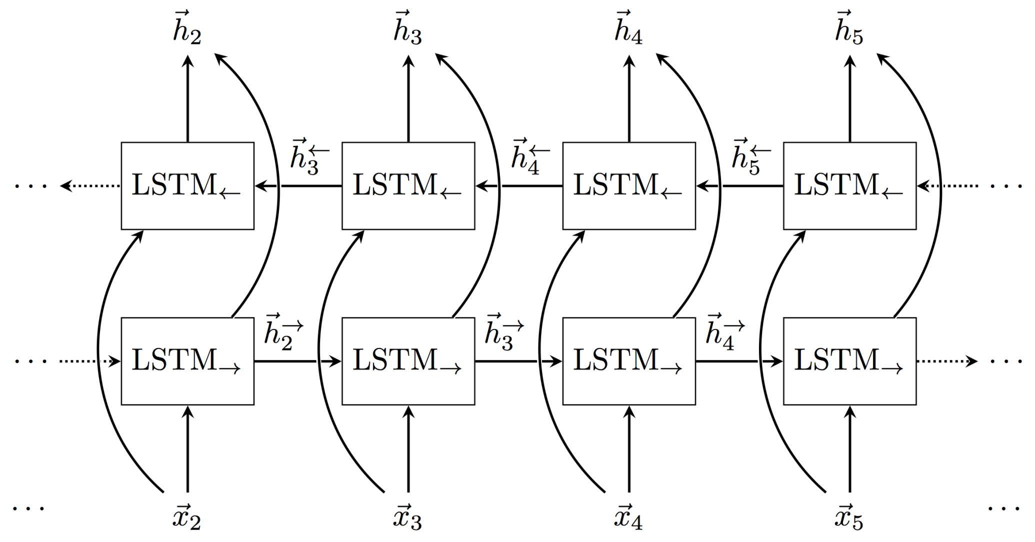


Figure 2.8: A diagram representing a BiLSTM RNN, processing the input sequence simultaneously forwards and backwards (to exploit both directions of temporal dependence) [Sak et al., 2014]

and future input is needed. Intuitively, in those tasks, both the previous words and the following words are important. Sometimes the information crucial for predicting the word w_t is at h'_t, a few time steps away, and $|t' - t|$ might be large. For example, when a model tries to predict the part-of-speech of *address*, it needs information from both the prior and following words, and some of which are quite far away. Recall that in the context *The address is Flat B, A Road, London, UK.* the word *address* is a noun, while if it is in context *I will address a speech.*, then the word *address* is a verb. In both cases, both the past and following words and LSTM mechanism are necessary to make a correct prediction.

Typical applications of BiLSTM include speech recognition, translation, handwritten recognition, protein structure prediction and part-of-speech tagging (see [Graves et al., 2013]).

2.2 NMT for GEC

2.2.1 Sequence-to-sequence Model

The RNN-seq2seq model was first introduced in
[Cho et al., 2014, Sutskever et al., 2014] and consists of two different RNNs: one was used as the encoder, and the other was used as the decoder. It is an end-to-end approach to sequence learning that generates a sequence of text from the input text. Sometimes, the input sequence and the output sequence are from different domains (e.g. the input sentences are in English, while the output sentences are the same sentences in French).

In our project, a seq2seq model consisting of two RNNs (the encoder and decoder) was used. The RNN encoder reads an input sequence and outputs a vector, and the decoder reads that vector to produce an output sequence. The encoder-decoder structure is presented in Figure 2.9. Unlike sequence prediction with a single RNN, where every input corresponds to an output, the seq2seq model frees us from constraints imposed by sequence length and order, which makes it ideal for GEC as the length of the input and the output might be different.

The RNN encoder reads an entire source sentence X into a single intermediate vector c:

$$c = q(h_T), \tag{2.19}$$

where q is a non-linear function, and H_t is the final hidden state of the encoder. The RNN decoder then outputs a translation Y by predicting the next word y_t based on the intermediate vector c and all the previously predicted words $\{y_1, y_2, ..., y_{t-q}\}$:

$$P(Y) = \Pi_{t=1}^{T'} P(y_t | \{y_1, y_2, ..., y_{t-1}\}, c_t). \tag{2.20}$$

The conditional probability $P(y_t | \{y_1, y_2, ..., y_{t-1}\}, c)$ is defined as:

$$P(y_t | \{y_1, y_2, ..., y_{t-1}\}, c) = g(s_t) = g(s_t), \tag{2.21}$$

where g is a non-linear function that outputs the probability of y_t. The decoder hidden state s_t at the time stop t is defined as:

$$s_t = f(s_{t-1}, y_{y-1}, c). \tag{2.22}$$

From 2.22, we can see that, the decoder hidden state s_t is different from the encoder hidden state h_t, as s_t depends on the previous hidden state s_{t-1}, previous output word y_{t_1} and the intermediate vector c.

Mathematically, given a sequence of inputs $(x_1, ..., x_T)$, a standard RNN computes a sequence of hypotheses $(y_1, ..., y_T)$ by iterating the following equations [Sutskever et al., 2014]:

$$h_t = sigm(W^{hx}x_t + W^{hh}h_{t-1}). \tag{2.23}$$

$$y_t = W^{yh}h_t. \tag{2.24}$$

In the LSTM-based GEC, given a source sentence with grammatical error $X = x_1, x_2, ..., x_m$, and a corrected target sentence $Y = y_1, y_2, ..., y_n$, the NMT models the conditional probability of *translating* the source sentence X to the target sentence Y as $p(y_1, y_2, ..., y_n|x_1, x_2, ..., x_m)$.

Then, the NMT system is trained to maximise log-likelihood:

$$\max_\theta \sum_{n=1}^{N} logP(Y^n|X^n, \theta) = \max_\theta \sum_{n=1}^{N} \sum_{n=1}^{T'} logP(y_t^n|y_1^n, y_2^n, ..., y_{t-1}^n, X^n, \theta), \tag{2.25}$$

where $\theta = [\theta_{encoder}, \theta_{decoder}]$ represents all parameters, N is the total number of training examples in the corpus and (X^n, Y^n) is the n^{th} pair.

2.2.2 Encoder-decoder Framework

Our NMT for GEC applies an encoder-decoder framework. A typical encoder-decoder structure is shown in Figure 2.9 for the purposes of illustration.

Encoder

In our work, the encoder is a BiLSTM RNN that reads an input sequence and creates hidden states, and a single vector which, in the ideal case, encodes the *meaning* of the input sequence into a single vector.

Usually, the encoder can be RNN, bidirectional RNN (BRNN) [Luong et al., 2015], transformer ([Chen et al., 2018]) or a convolutional neu-

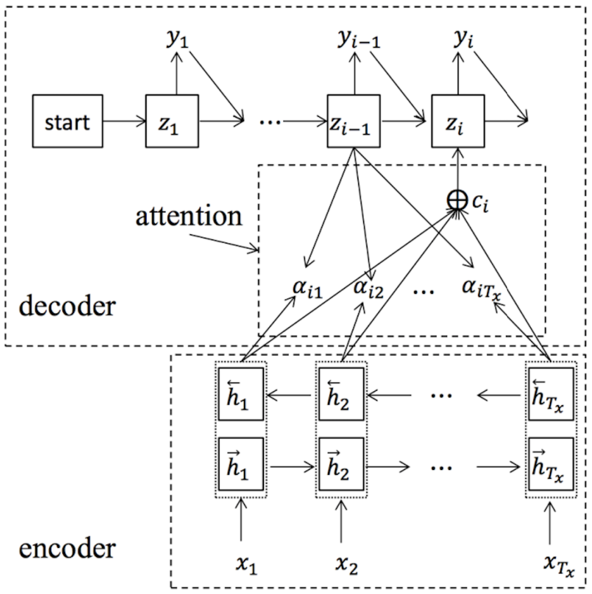


Figure 2.9: Attention-based encoder-decoder framework of our NMT for GEC, where $X = (x_1, x_2, ..., x_{Tx})$ is the word embedding of the source sentence; $h = (h_1, h_2, ..., h_{Tx})$ is the encoder hidden states; $Y = (y_1, y_2, ..., y_i)$ is the predicted word; and the c_i is the context vector that encodes the connection between the i^{th} decoder hidden state and the source sentence [Zhang and Zong, 2016]

ral network (CNN) ([Meng et al., 2015]). In our project, the BRNN encoder is used, which consists of two independent encoders: one encoding the forward sequence and the other the backward sequence. The output and final states are concatenated.

Decoder

The decoder takes the vector from the encoder output vectors and outputs a sequence of words to create the translation: the decoder at step t is using the reference token for step $t-1$ to calculate the new output. In our model, the attention mechanism allows the decoder network to *focus* on a different part of the encoder output for every step of the decoder output. The attention mechanism is introduced in Section 2.2.3.

Usually, the decoder can be an RNN [Yuan and Briscoe, 2016], a transformer [Gu et al., 2017], or a convolutional neural network (CNN) [Klein et al., 2017]. In our project, an LSTM RNN is used. Additionally, it applies attention to the source sequence and implements input feeding.

2.2.3 Attention Mechanism

When a human translates a sentence, he or she pays particular attention to the word he or she is presently translating. Neural networks can achieve this same behaviour using attention, i.e. by focusing on a subset of the information it is given. For example, an RNN can attend to the output of another RNN. At every time step, it focuses on different positions in the other RNN [Bahdanau et al., 2014].

An attention mechanism establishes direct short-cut connections between the target and the source by paying attention to the relevant source content. The attention matrix is an alignment matrix between the source and reference sentences, showing the mapping and correspondence of the words. Global attention takes a matrix and a query vector. Then, it computes a parameterised convex combination of the matrix based on the input query.

When the attention mechanism is used, the model calculates a set of attention weights, which will be multiplied by the encoder output vectors to create a weighted combination. The result (called attn_applied in the code) should contain information about that specific part of the input sequence, and thus help the decoder choose the grammatically correct output words.

An example of the alignment matrix between the source and target sentences for NMT is shown in Figure 2.10, where the global attention takes a matrix and a query vector and then computes a parameterised convex combination of the matrix based on the input query.

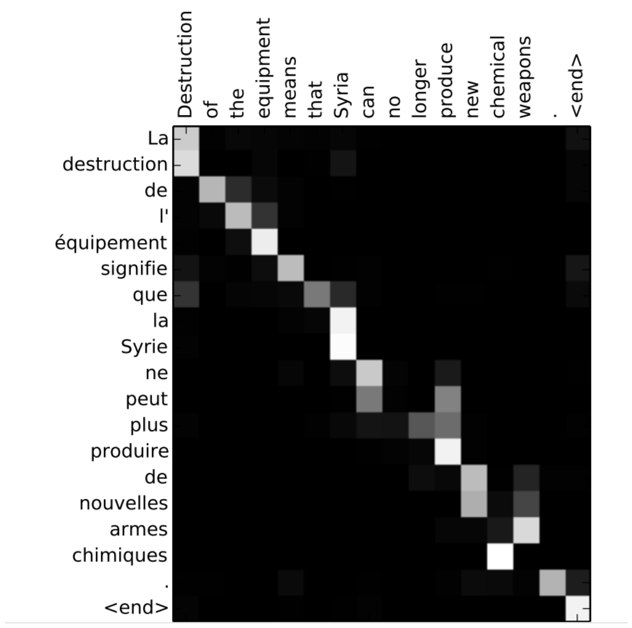


Figure 2.10: Example attention alignments. The x-axis and y-axis correspond to the words in the source sentence (in English) and the predicted translation (in French) respectively. Each individual square visualises the weight α_{ij} of the j^{th} source word for the i^{th} target word, where the darker the colour is, the heavier the weight is [Bahdanau et al., 2014]

Calculating the attention weights is accomplished using another feed-forward layer attention, using the decoder's input and hidden state as inputs. The attention weights can be summarised by the equations below:

$$\alpha_{ts} = \frac{exp(score(\mathbf{h}_t, \overline{\mathbf{h}}_s))}{\sum_{S}^{s'=1} exp(score(\mathbf{h}_t, \overline{\mathbf{h}}_s))}. \tag{2.26}$$

Based on the attention weights, the system computes a context vector as the weighted average of the source states. The system combines this context vector with the current target hidden state to calculate the final attention vector. The attention vector is then fed as an input to the next time step. An attention-based encoder-decoder system for machine translation is presented in Figure 2.9 for illustrative purposes:

The context vector is:

$$c_t = \sum_s \alpha_{ts} \overline{\mathbf{h}}_s. \tag{2.27}$$

Finally, the attention vector is:

$$a_t = f(c_t, h_t) = \tanh(\mathbf{W}_c[\mathbf{c}_t; \mathbf{h}_t]). \tag{2.28}$$

$$score(\mathbf{h}_t, \overline{\mathbf{h}}_s) = \mathbf{h}_t^T \mathbf{W} \overline{\mathbf{h}}_s), \tag{2.29}$$

where the ***score*** function is used to compared the target hidden state, h_t with each of the source hidden states, $\overline{h}_s$. The result is normalised to produce attention weights (a distribution over source positions). There are different forms of score functions. In our project, the function in [Luong et al., 2015] was used. Paper [Luong et al., 2017] found that it is important to feed the attention vector to the next time step to inform the network about past attention decisions, as demonstrated.

Finally, the RNN decoder outputs a translation Y:

$$P(Y) = \Pi_{t=1}^{T'} P(y_t | \{y_1, y_2, ..., y_{t-1}\}, c_t). \tag{2.30}$$

Empirically, when the attention mechanism is added, the performance of the GEC system is improved [Yuan and Briscoe, 2016]. The attention-based model improves the accuracy by focusing on the important words. It is particularly important that the GEC model can learn the grammatically correct word order. An example is as follows:

> **Example 2.2.1** ***Incorrect word order:***
>
> *Incorrect: I was in the shopping mall **shopping**.*
>
> *Correct: I was **shopping** in the shopping mall.*

In this thesis, we applied the attention API of PyTorch seq2seq to the encoder where the current target hidden state is compared with all source states in order to derive attention weights.

2.3 Dataset

Data collected from renowned, standardised English test services such as the International English Language Testing System (IELTS), First Certificate in English (FCE), Cambridge English Qualifications Business (BEC), Test of English as a Foreign Language (TOEFL), and Test of English for International Communication (TOEIC), are often used in academia. The nature of these English tests justify the collected data sets as comprehensive resources both research and also for learners of English.

The carefully designed exam questions bring together definite known grammatically correct answers along with repeated errors in the application of different grammatical rules introduced by different examinees. Compared to real-world data gathered in daily software use, the English test data sets are more widely recognised in academia; moreover, they are generally considered more reliable and precise, better organised and excellent for uniform performance metrics, and thus more orthodox.

In general, there are two broad categories of parallel data for GEC. The first is error-coded text, in which annotators have coded spans of learner text containing an error. The second class of GEC corpora are parallel datasets, which contain the original text and a corrected version of the text.

We use the publicly available *FCE parallel dataset*, which is a part of the CLC [Nicholls, 2003]. It was released in 2011 in [Yannakoudakis et al., 2011]. The FCE dataset contains 1,244 scripts produced by learners taking the FCE examination between 2000 and 2001. There are two essays in each script, including reports, narration writing and expository writing. The lengths of the sentences are between 2 and 133 words, with an average length is 31. The target vocabulary size is 11,281, and the source vocabulary size is 14,443. The FCE dataset contains about half a million words and more than $50k$ errors [Yuan, 2017]. An example sentence pair is provided below:

Example 2.3.1 ***FCE Dataset***

Source: I am sad to read about Richard not being at his best .

Target: I am sad to read about Richard not being well .

The FCE texts have been manually annotated by linguists at the University of Cambridge using a taxonomy of approximately 80 error types. The publicly available FCE dataset contains 28,976 pairs of parallel sentences (the source and the corrected sentence pairs) for training (898×10^3 tokens on the target side), 2,242 pairs of parallel sentences for validation (34,627 tokens on the target side), and 2,757 pairs of parallel sentences for testing (85×10^3 tokens on the target side).

2.4 Evaluation Metrics

When evaluating a GEC system, the output hypotheses (see Section 3.1.3 for details) are compared to gold-standard references provided by human linguists.

An ideal evaluation method for GEC task rewards the error being appropriately corrected; captures the difference between *the error is not changed* and *there are wrong edits*; and is accurate at both the corpus-level and sentence-level.

We chose to evaluate the system performance using two automatic evaluation metrics: $M^2(F_{0.5})$ score [Dahlmeier and Ng, 2012] and GLEU score [Napoles et al., 2015]. The $M^2(F_{0.5})$ score was the official score used in the 2013 and 2014 CoNLL shared tasks (see [Kao et al., 2013, Ng et al., 2014]). The GLEU score is a simple variant of Bilingual Evaluation Understudy (BLEU) in [Papineni et al., 2002], that better correlation with human judgements on the CoNLL 2014 shared task test set [Ng et al., 2014].

When we chose the evaluation scores, we considered both whether these methods logically and mathematically fit with the GEC task, and whether these methods agreed with the golden human-evaluation.
In [Napoles et al., 2015], they conducted the first human evaluation of GEC system hypotheses and showed that the rankings produced by GLEU score agree with human's judgement the most among BLEU score, M^2 score and GLEU score in the GEC tasks.

Analysis of the $M^2(F_{0.5})$ score, BLEU score and GLEU score presented below.

2.4.1 BLEU Score

The BLEU score was first introduced in
[Papineni et al., 2002] and is now used as the dominant method for machine translation evaluation. In machine translation between different languages, the BLEU score successfully estimates the quality of the text produced by

machine translation systems and agrees with the evaluation of both monolinguals' and bilinguals' [Papineni et al., 2002].

Firstly, it uses a modified n-gram precision (p_n) to compare a hypothesis against multiple references:

$$p_n = \frac{\sum_{n-gram \in C} count_{clip}(n-gram)}{\sum_{n-gram \in C} count(n-gram)}, \tag{2.31}$$

where C is a hypothesis sentence. The count of each n-gram in C is clipped by its maximum reference count observed in any single reference for that n-gram:

$$count_{clip} = min(count, max\_ref\_count). \tag{2.32}$$

Calculation of BLEU score proceeds as follows:

$$BP = \begin{cases} 1 & \texttt{if } c > r \\ e^{1-r/c} & \texttt{if } c \leq r \end{cases} \tag{2.33}$$

$$BLEU\_score = BP \cdot \exp(\sum_{n=1}^{N} w_n \log p_n), \tag{2.34}$$

where p_n is the geometric average of the modified n-gram precisions, using n-grams up to length N and positive weights w_n summing to one; the c is the length of the predicted sentence; while r is the length of the target sentence; and the BP is the brevity penalty,

The disadvantage of using BLEU scores in GEC tasks is that, in machine translation, an untranslated word is highly possible an error, as it means that the system has failed in translation. Therefore, the BLEU score would be low if the hypothesis sentence is similar to the input sentence. However, this is not the case in GEC, as most of the words are not touched in the translation.

Moreover, a BLEU score is intended to apply only to the document rather than to individual sentences, and hence, its application to individual sentences is inaccurate and correlates poorly with human judgements in both GEC and machine translation [Mutton et al., 2007].

Due to the above disadvantages, BLEU score has negative correlations with human rankings, suggesting that it is not a suitable metric for judging the results of a GEC task [Napoles et al., 2015]. The correlation of metrics with the gold-standard human ranking is calculated using Pearson's r and Spearman's ρ. The Pearson's r score for the BLEU score is -0.125, and the Spearman's ρ score is -0.225.

Considering the above disadvantages, we elected not to use BLEU score as the evaluation method for our GEC task.

2.4.2 GLEU Score

GLEU score, introduced in [Napoles et al., 2015], is a simple variant of BLEU score for GEC that takes the source into account. Similar to the BLEU score, the GLEU score can work with multiple references at the corpus-level. It can be used as a generic evaluation method independent of the annotation scheme but fails to provide detailed system performance for each error type.

The GLEU score modifies the n-gram precision in BLEU to assign extra weight to n-grams present in the hypothesis that overlap with the reference but not the source ($R\backslash S$) and to penalise those in the hypothesis that are in the source but not the reference ($S\backslash R$). For a correction hypothesis C with a corresponding source S and reference R, the modified n-gram precision (p'_n) for a GLEU score is:

$$p_n = \frac{\sum_{n-gram \in C} count_{R\backslash S}(n-gram) - \lambda(count_{S\backslash R}(n-gram)) + courn t_R(n-gram)}{\sum_{n-gram \in C} count_S(n-gram) + \sum_{n-gram \in R\backslash S} count_{R\backslash S}(n-gram)}, \tag{2.35}$$

where the weight λ determines the extent to incorrectly changed n-grams are penalised. Given a bag of n-grams B, the counts in 2.35 are calculated as:

$$count_B(n-gram) = \sum_{n-gram' \in B} d(n-grams, n-gram'). \tag{2.36}$$

$$d(n-gram, n-gram') = \begin{cases} 1 & \texttt{if } n-gram = n-gram' \\ 0 & \texttt{otherwise} \end{cases} \tag{2.37}$$

By updating the n-gram precision, the GLEU score is as follows:

$$GLEU(C, R, S) = BP \cdot exp(\sum_{n=1}^{N} w_n logp'_n). \quad (2.38)$$

Recall that BLEU score would be low if the hypothesis sentence is similar to the input sentence. This observation motivates a small change to BLEU that takes the source sentence into account as well and computes n-gram precisions over the reference but assigns more weight to n-grams that have been correctly changed from the source. Therefore, GLEU score rewards corrections while also correctly crediting unchanged source text.

Moreover, GLEU score is not just intended to apply only to the document instead of sentences, but also apply to individual sentences. Therefore, sentence-level GLEU score is accurate and correlates with human judgements in both GEC and machine translation [Mutton et al., 2007]. At corpus-level GLEU score, instead of averaging the sentence-level scores, GLEU score combines first all numerators and denominators, and then the final aggregation is computed.

The correlation of metrics with the golden human ranking is calculated with Pearson's r and Spearman's ρ. The GLEU score is the best among GLEU score, $M^2(F_{0.5})$ score, and BLEU score. The Pearson's r score is 0.542, and the Spearman's ρ score is 0.555.)

Considering the above advantages compare with BLEU score, we elected to use the GLEU score instead of the BLEU score as the evaluation method.

Using GLEU score as the Loss-function in GEC

Currently, NMT models are trained to maximise likelihood; however, the primary evaluation score is now a GLEU score. Hence, there is a mismatch between the training and evaluation criteria. To ensure better results, we proposed to train the NMT models by maximising GLEU score instead; however, this method was not adopted, and it can be analysed as follows:

Recall that the formula of GLEU score is $GLEU = BP \cdot exp(\sum_{n=1}^{N} w_n logp'_n)$, such that the vector representations of the token is not smooth and hence are not differentiable. Extra efforts are needed to make the GLEU score differentiable. Additionally, [Casas et al., 2018] states that models using GLEU score as the loss function are deficient in seq2seq tasks, suggesting that the GLEU score would not be a good training loss for our GEC task. Following analysis, they stated that using a GLEU score as the loss leads the trained model to generate an averaged vector in token space, instead of truly learning the error correction.

Considering the complexity, workload and reported weak results associated with this task, GLEU score was not used as the loss function in this project.

2.4.3 M^2 Score

The M^2 scorer, introduced in [Dahlmeier and Ng, 2012], is used to evaluate system performance according to how well its proposed corrections match the gold-standard text. It computes the sequence of phrase-level edits between a source sentence and a system's hypothesis that achieves the highest overlap with the gold standard annotation. The system is evaluated by computing precision (P), recall (R) and F-score. The calculations are as follows:

$$P = \frac{\sum_{i=1}^{n} |\mathbf{e}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{e}_i|}. \tag{2.39}$$

$$R = \frac{\sum_{i=1}^{n} |\mathbf{e}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{g}_i|}. \tag{2.40}$$

$$F_\beta = (1+\beta^2) \times \frac{P \times R}{(1+\beta^2) \times P + R}, \tag{2.41}$$

where e is the system edits $(e1, ..., en)$; g is the set of gold edits $(g1, ..., gn)$; β is a constant (in our project, the β is 0.5); and $e_i \cap g_i$ is the intersection between e_i and g_i as:

$$e_i \cap g_i = \{e \in e_i | \exists g \in g_i(match(e,g))\}. \tag{2.42}$$

The most common F-scores are the F_1 score, which weights P and R evenly, and the $F_{0.5}$ score, which weights P twice as much as R:

$$F_1 = 2 \times \frac{P \times R}{P + R}. \tag{2.43}$$

$$M^2(F_{0.5}) = (1+0.5^2) \times \frac{P \times R}{(1+0.5^2) \times P + R}. \tag{2.44}$$

According to [Yuan, 2017], when building GEC systems, minimising the number of unnecessary corrections is often regarded as more important than covering all the errors, as the users would want the system to make accurate corrections while covering all errors is not necessary for them. That is to say, high P is often more preferred over high R.

The advantage of $M^2(F_{0.5})$ score compared with the traditional F-score is that $M^2(F_{0.5})$ score can capture phrase-level edits. Secondly, the $M^2(F_{0.5})$ score is accurate at both corpus-level and sentence-level. Also, $M^2(F_{0.5})$ score successfully shows how many errors are spotted and corrected.

However, the $M^2(F_{0.5})$ score has disadvantages as well. Firstly, phrase-level edits can be inaccurate because the lattice treats a long phrase deletion as one edit. Secondly, the F-score does not capture the difference between *error not changed* and the incorrect editing of correct words, i.e., *wrong edits*.

Unfortunately, the $M^2(F_{0.5})$ score only correlates moderately with human rankings, suggesting that it is not an ideal metric for judging the results of a GEC task [Napoles et al., 2015]. The correlation of metrics with the gold-standard human ranking is calculated using Pearson's r and Spearman's ρ. The Pearson's r score of the $M^2(F_{0.5})$ score is 0.358, and the Spearman's ρ score of the $M^2(F_{0.5})$ score is 0.429.

One of the reasons that the $M^2(F_{0.5})$ score does not correlate with human evaluation is that $M^2(F_{0.5})$ does not provide an indicator of improvement on the original text, therefore, it is not possible to compare GEC systems against a *do-nothing* baseline that keeps the input text unchanged. They $M^2(F_{0.5})$ score of the *do-nothing* baseline is 0 and therefore, an increase in P, R or F-score does not necessarily mean a reduction in the actual error rate.

Despite the disadvantages and moderate correlation with human ranking, we chose to report the $M^2(F_{0.5})$ scores as it is one of the standard methods, and was used to rank error correction systems in the 2013 (F_1) and 2014 Conference on Computational Natural Language Learning (CoNLL) ($F_{0.5}$) shared tasks. Using the $M^2(F_{0.5})$ score allows us to compare the results of our work with others, and can adequately demonstrate whether the hypothesis edits match the gold-standard.

Chapter 3

Baseline

3.1 Pipeline

We trained a baseline NMT model that can correct the grammatical errors in the input text using OpenNMT-py[1], a Python generic deep-learning framework on top of PyTorch that provides a flexible and efficient way to implement complex NMT models. The procedure of training our GEC system consists of three steps: 1) data preprocessing; 2) model training; and 3) model testing. The details of the three steps are as follows:

3.1.1 Step 1: Preprocessing

Firstly, we tokenise the text by splitting the corpus into space-separated tokens. Then, the system builds dictionaries for the source text and the target text separately. These dictionaries are not merged because the source text is likely to contain misspelt words and hence has a more extensive vocabulary. In the dictionary, each word is allocated with an index as its identifier. We refer to the number of the words each sentence as *the length of the sentence.* On average, the sentence length in the dataset (including training set, validation set and training set) is 31. The target vocabulary size is 11,281, and the source vocabulary size is 14,443. Finally, the data preprocessing passes over the tokenised training and validation corpus, to generate the sequences of indices used by the training system.

[1]https://github.com/OpenNMT/OpenNMT-py

3.1.2 Step 2: Model Training

The baseline model is a word-level attention-based RNN seq2seq NMT GEC model, which was introduced in Section 2.2. The baseline model was built based on the model in [Klein et al., 2017].

During training, the word embeddings are learned. Initially, each word is assigned to a random vector, after which the vectors are jointly learnt together with the model. For logging and tracking performance, word perplexity (ppl), defined below, is displayed every 50 iterations.

$$ppl(X, Y) = \exp \frac{-\sum_{i=1}^{|Y|} \log P(y_i | y_{i-1}, ..., y_1, X)}{|Y|}, \tag{3.1}$$

where X is the source sequence, Y is the target sequence for reference, and y_i is the i^{th} target word. The numerator is the negative log-likelihood, i.e., loss function value. The ppl loss shows how well our model fits unseen data, and hence, we want the value of ppl to decrease, which will indicate that our model fits the training data well. At the end of an epoch, the logs report the GLEU score and the validation perplexity of the validation data.

In our project, all models are trained on Azure Standard NC6[2]. During training, the parameters are optimised by the Adam optimiser with a decay mechanism (see Section 3.2.1 for the details of the Adam optimiser). In total, $21,827,425$ parameters were learnt, including $7,052,644$ parameters for the encoder, and $14,774,781$ parameters for the decoder. The initial learning rate drops by 50% from 0.001 if the likelihood does not improve for two sequential epochs.

After two hours, the loss begins to converge at the 12^{th} epoch. If the GLEU score on the validation set does not improve for five sequential epochs, the training will stop automatically (i.e. the early stopping method to avoid overfitting). Otherwise, after 15 epochs, we select the model with the highest GLEU score as the final trained output model.

3.1.3 Step 3: Testing

To test the trained model, we run the trained model on the FCE test set and collect hypotheses, which are the results from the beam search as introduced

[2]six CPU cores, Intel E5-2690v3, 56 GB RAM, with GPU acceleration (Nvidia Tesla K80, 4992 cores), CUDA 9.1

below. The hypotheses are then compared with the ground-truth, after which the GLEU score and the $M^2(F_{0.5})$ score are reported.

Beam Search We use a beam search to find a correction that approximately maximises the likelihood. In decoding, the beam search is applied to the target words space but not the feature space. When the beam path is complete, the associated features are selected along the best path. To trade-off translation time and search accuracy, we use a beam size of 5 (see [Klein et al., 2017]). Despite the fact that beam search can provide us with the n-best hypotheses, we only use the 1-best hypothesis.

3.2 Hyper-parameters

We used the same hyper-parameters as in [Klein et al., 2017]. The key hyper-parameters of the model are shown in Table 3.1 below:

Table 3.1: Key Hyper-parameters of the Baseline Model

Parameter	**Value**
RNN size	512
word vector size	300
dropout	0.30
optimiser	Adam
loss function	negative log likelihood loss
beam size	5
maximum batch size	64
epoch	15
initial learning rate	0.001

3.2.1 Adam Optimiser

In the baseline, stochastic gradient descent (SGD) optimiser, a stochastic approximation of the gradient descent optimisation, was used at first. The initial learning rate was 1.0 and would drop by 50% if the likelihood did not improve (see [LeCun et al., 1998]).

Empirical results demonstrate that the Adaptive Moment Estimation (Adam) optimiser works well in practice and compares favourably to other stochastic optimisation methods [Kingma and Ba, 2014]. Therefore, we would expect that using the Adam optimiser would improve the GEC results.

When other parameters are the same (see Table 3.1), the GLEU score of the model that uses the SGD optimiser is 65.19, while the model that uses the Adam optimiser scores 66.90. Additionally, the model that uses Adam optimiser converges three epochs faster than its counterpart (12^{th} v.s. 15^{th}). Therefore, the Adam optimiser is used in both the baseline and subsequent research. The details of the Adam optimiser is as follows:

The Adam optimiser is an update to the root mean square property (RMSProp) optimiser [Tieleman and Hinton, 2012]. In this optimisation algorithm, running averages of both the gradients and the second moments of the gradients are used, i.e., helping the model to escape local maximum. This is an algorithm designed for first-order gradient-based optimisation of stochastic objective functions, based on adaptive estimates of lower-order moments.

Given parameters $w^{(t)}$ and a loss function $L^{(t)}$, where t indexes the current training iteration, Adam's parameter update is given by:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}. \tag{3.2}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2. \tag{3.3}$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^t}. \tag{3.4}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^t}. \tag{3.5}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta\frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon}, \tag{3.6}$$

where ϵ is a small number used to improve numeral stability (division by 0); β_1 and β_2 are the forgetting factors for gradients and second moments of gradients, respectively; g_t is the gradients with respect to stochastic objective at time step t; m_t is the biased first moment estimate; u_t is the exponentially weighted infinity norm; and θ represents the parameters

[Kingma and Ba, 2014].

3.2.2 Regularisation

To prevent neural networks from overfitting and increase generalisation capacity, dropout is used in our thesis. This is an effective and straightforward regularisation technique applied during training (see [Srivastava et al., 2015]). The idea behind the technique is to disable a random batch of individual neurons with a given probability. It can reduce overfitting in RNN by preventing complex co-adaptations on training data. In our thesis, we set the dropout probability to 0.30 at the penultimate layer to facilitate performance gain. Figure 3.1 presents a depiction of the dropout technique.

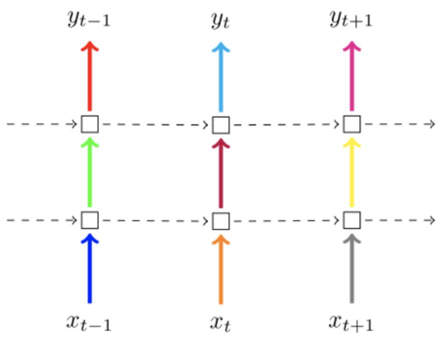


Figure 3.1: Dropout applied on RNNs. Each square is an RNN unit, where horizontal arrows are the time steps; vertical arrows are the input and output of each RNN unit; the coloured arrows represent the dropped-out inputs, with different colours corresponding to different dropout rates. Notably, the dash lines are the standard input and output with no dropout. [Gal and Ghahramani, 2016]

3.3 Handling Unknown Words

The input text will likely contain unknown words including unseen misspelt words, new words and foreign words.

Due to a lack of accurate information, in the baseline, all unknown words are marked with UNK (unknown) tokens in the encode. The system then replaces the UNK tokens with the source token that had highest attention weight (may or may not the original word).

Despite using UNK token replacement technology, the hypotheses of the baseline model would still edit some grammatically correct OOV words, which is grammatically incorrect by changing them to other undesirable words. In Chapter 4, we applied a copy mechanism to avoid grammatically incorrect edits, and the results show a considerable improvement (see Section 4.1.1 for details).

3.4 Negative Log-likelihood Loss Function

In this project, we used the negative log-likelihood (nll) as the training objective, and the reason is as follows. At each time step t, the decoder outputs a probability over the words in the dictionary we built. Our goal is to match the output word to the target word, which is similar to an N-class classification task where N is the vocabulary size. In N-class classification tasks, nll is often used as the loss function (see [Platt et al., 1999]).

The input given through a forward call is expected to contain log-probabilities of each class, where the input size is (N, C), such that N is the batch size, and C is the vocabulary size. Obtaining log-probabilities in a neural network is easily achieved by adding a *LogSoftmax* layer in the last layer of our network. The losses are summed for each sentence, after which the losses of all output sentences are averaged in the batch. The loss can, therefore, be described as:

$$l(x, y) = \sum_{n=1}^{N} \frac{l_n}{\sum_{n=1}^{N} w_{y_n}}, \tag{3.7}$$

where x is the input, y is the target, N is the batch size, $l(x, y)$ is the loss and w is the weight.

3.5 Baseline Performance

As shown in Table 3.2, the GLEU score of our baseline system is 66.90, and the $M^2(F_{0.5})$ score is 15.71 (tested on the FCE test set). We, therefore, see that the $M^2(F_{0.5})$ score is higher than 0, which means that the GEC system successfully corrected some errors. However, the GLEU score of the source is 74.79, higher than the baseline, suggesting that a considerable number of grammatically correct words have been improperly edited. In comparison, the state-of-the-art is the model in [Yannakoudakis et al., 2017], which achieved a GLEU score of 71.76 and the $M^2(F_{0.5})$ score of 55.60. Hence further improvement is needed.

Table 3.2: Performance of the Baseline Model

Model	**GLEU Score**	$M^2(F_{0.5})$ **Score**
baseline	66.90	15.71
Yannakoudakis	71.16	55.60

Additionally, we visualised the attention matrices in Figure 3.2 in order to illustrate the alignments between the source sentences and the output hypothesis sentences. There is no numerical evaluation score for attention matrices. Therefore, we performed qualitative evaluation by manually printing and plotting twenty attention matrices as a random selective examination. We found that sixteen of the models paid attention to the correct source words when *translating*. From the randomly selected attention matrices, we believe that the attention mechanism works well, as their alignments are correct. An example that shows that the alignment is correct is presented in Figure 3.2.

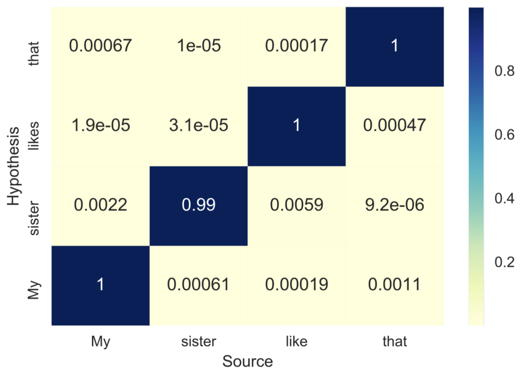


Figure 3.2: An example of the sentence alignment in our GEC task. The x-axis and y-axis correspond to the words in the source sentence and the generated translation respectively. Each pixel shows the weight α_{ij} of the annotation of the j^{th} source word for the i^{th} target word. A darker pixel represents a larger attention weight.

Chapter 4

Improvement

Recall that in Section 1.1.3 we mentioned that the current attention-based NMT systems for GEC could be further improved if we apply data augmentation (adding information of PoS tags, GED or semantics) and model improvement (using copy mechanism). We can also try a combination of the individual improvements.

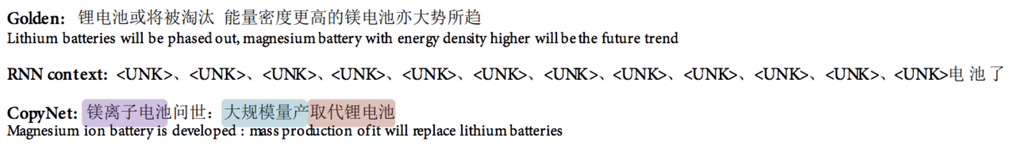
4.1 Individual Improvements

4.1.1 COPYNET Copying Mechanism

Motivation

We explored the incorporation of the COPYNET Copying Mechanism into our NMT GEC task to avoid incorrectly rectifying words that are already correct.

The results of the baseline model show that the model would sometimes incorrectly rectify words that are already correct. For example, the sentence *That is a **hospital**.* has no grammatical errors and is correct. However, the word ***hospital***, was rectified to ***university***, because the sentence *This is a university*, is a target sentence in the training set, i.e., my model is overfit on the phrase *This is a **university***. By this, it means that no matter what the final hidden state from the encoder is, when the decoder generates *this is a*, the next word with the highest conditional probability is always ***university***.



Golden: 锂电池或将被淘汰 能量密度更高的镁电池亦大势所趋
Lithium batteries will be phased out, magnesium battery with energy density higher will be the future trend

RNN context: <UNK>、<UNK>、<UNK>、<UNK>、<UNK>、<UNK>、<UNK>、<UNK>、<UNK>、<UNK>、<UNK>、<UNK>电 池 了

CopyNet: 镁离子电池问世：大规模量产取代锂电池
Magnesium ion battery is developed : mass production of it will replace lithium batteries

Figure 4.1: Examples of COPYNET correctly directly copying OOV words in a Chinese summarisation task, where the OOV words are underlined; the words with a higher copy mode probability than the generate mode probability are highlighted [Gu et al., 2016].

Typically, in tasks like GEC, dialogue-generating and text summarising, certain words in the input sentences are selectively replicated in the target hypotheses, and therefore, a new mechanism is needed in order to decide when to operate. In the GEC task, taking the source words into account during the prediction step might avoid rectifying words that are already correct. Therefore, in this task, we would expect that using COPYNET mechanism would improve the GEC system. An example of COPYNET correctly copying OOV words in a Chinese summarisation task is shown in Figure 4.1.

COPYNET Copying Mechanism

COPYNET is a mechanism first introduced in [Gu et al., 2016] that incorporates copying into neural network-based seq2seq learning. The mechanism can not only perform the standard seq2seq generation of words, but also copying certain words in the input sequence correctly. In detail, COPYNET can efficiently integrate the regular technique of word generation into the decoder with the new copying mechanism which can choose sub-sequences in the input sequence and put them at the proper places in the output sequence. Technically, COPYNET is still an encoder-decoder, but in a slightly more generalised sense. As illustrated in Figure 4.2, the source sequence is transformed by the encoder, and then fed into the decoder to generate the target sequence. The details are as follows:

Encoder: The same as in the baseline([Bahdanau et al., 2014]). A bidirectional RNN is used to transform the source sequence into a series of hidden states with the equal length, with each hidden state h_t corresponding to word x_t. This new representation of the source, $y = h_1, ..., h_{T_S}$, is considered to be a short-term memory, which will later be accessed in multiple ways when generating the target sequence (decoding).

Decoder: An RNN that reads M and predicts the target sequence. It

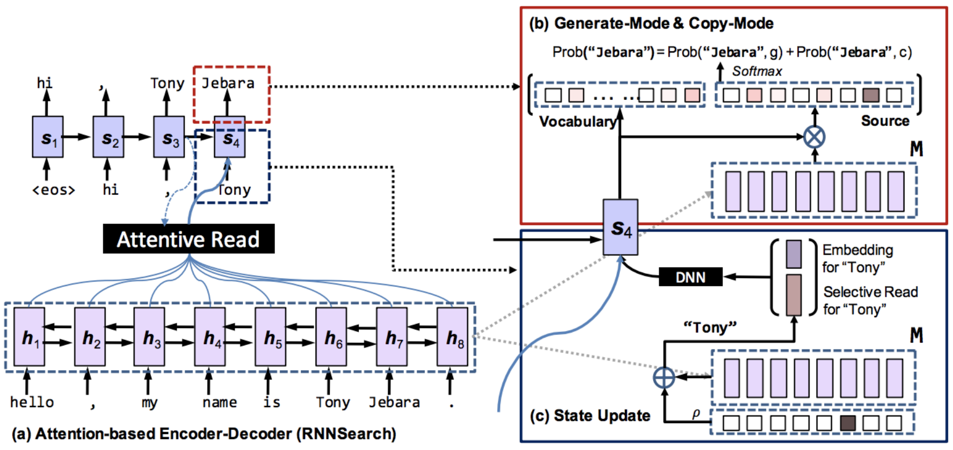

(b) Generate-Mode & Copy-Mode
Prob("Jebara") = Prob("Jebara", g) + Prob("Jebara", c)
Softmax
Vocabulary
Source
M
hi
,
Tony
Jebara
<eos>
hi
,
Tony
Attentive Read
Embedding for "Tony"
Selective Read for "Tony"
DNN
"Tony"
hello
,
my
name
is
Tony
Jebara
.
(a) Attention-based Encoder-Decoder (RNNSearch)
(c) State Update

Figure 4.2: The overall diagram of COPYNET, where the source sequence is transformed by the encoder, and then fed into the decoder to generate the target sequence. Note that in step (b), the probability distribution (softmax) is built on the union set of output dictionary and the words in the source sentence [Gu et al., 2016]

is similar to the canonical RNN-decoder used by [Bahdanau et al., 2014], although, with the following important differences:

1. Prediction: COPYNET predicts words based on a mixed probabilistic model of two modes - the generate-mode and the copy mode, where the latter picks words from the source sequence (this is further explained in later sections);
2. State Update: the predicted word at time $t-1$ is used in updating the state at t, but COPYNET uses not only its word-embedding but also its corresponding location-specific hidden state in M;
3. Reading M: in addition to the attentive read to M, COPYNET also has *selective read* to M, which leads to a effective hybrid of content-based and location-based addressing.

COPYNET [Gu et al., 2016] uses an extended dynamic dictionary that contains words in the target dictionary plus an arbitrary number of additional words introduced by the source sentence. For each source sentence, they have a source map that maps each source word to an index in the target dictionary if it is known, or else to an extra word. The copy generator is an extended version of the standard generator that computes three values

[Gu et al., 2016]:

1. p_{soft_max}: the standard softmax over target dictionary (tgt_{dict});
2. p_z: the probability of instead of copying a word from the source, computing it using a Bernoulli distribution;
3. p_{copy}: the probability of copying a word instead. Taken from the attention distribution directly.

The model returns a distribution over the extended dictionary (the union set of the output dictionary and the words in the source sentence), computed as:

$$p(w) = p(z = 1)p_{copy}(w) + p(z = 0)p_{soft\_max(w)}. \tag{4.1}$$

In detail, given the decoder RNN state s_t at time t together with M, the probability of generating any target word y_t, is given by the *mixture* of probabilities as follows:

$$p(y_t|\mathbf{s}_t, y_{t-1}, \mathbf{c}_t, \mathbf{M}) = p(y_t, g|\mathbf{s}_t, y_{t-1}, \mathbf{c}_t, \mathbf{M}) + p(y_t, \mathbf{c}|\mathbf{s}_t, y_{t-1}, \mathbf{c}_t, \mathbf{M}), \tag{4.2}$$

where g stands for the generate mode, and c the copy mode. The probability of the two modes are given respectively as:

$$p(y_t, g|\cdot) = \begin{cases} \frac{1}{Z}e^{\psi_g(y_t)} & y_t \in V \\ 0 & X \bigcap \overline{V} \\ \frac{1}{Z}e^{\psi_g(UNK)} & y_t \notin V \bigcup X \end{cases} \tag{4.3}$$

$$p(y_t, c|\cdot) = \begin{cases} \frac{1}{Z}\sum_{j:x_j=y_t} e^{\psi_c(x_j)} & y_t \in X \\ 0 & otherwise \end{cases} \tag{4.4}$$

where $\psi_g(\cdot)$ and $\psi_c(\cdot)$ are score functions for the generate mode and the copy mode, respectively, and Z is the normalisation term shared by the two modes, $Z = \sum_{v\in V \bigcup UNK} e^{\psi_g(v)} + \sum_{x\in X} e^{\psi_c(x)}$.

Generate-Mode The same scoring function as in the generic RNN encoder-decoder [Bahdanau et al., 2014] is used, i.e.

$$\psi_g(y_t = v_i) = \mathbf{v}_i^\top \mathbf{W}_o \mathbf{s}_t, v_i \in V \bigcup UNK, \tag{4.5}$$

where $\mathbf{W}o \in \mathbb{R}^{(N+1)\times d_s}$ and v_i is the one-hot indicator vector for v_i.

Copy-Mode The score for *copying* the word x_j is calculated as:

$$\psi_c(y_t = x_j) = \sigma(\mathbf{h}_j^\top \mathbf{W}_c)\mathbf{s}_t, x_j \in X, \tag{4.6}$$

where $W_c \in \mathbb{R}^{d_h \times d_s}$, and σ is a non-linear activation function, considering that the non-linear transformation in Equation 4.6 can help project s_t and $_j$ in the same semantic space.

The illustration of the decoding probability $p(y_t|\cdot)$ as a 4-class classifier is shown in Figure 4.3.

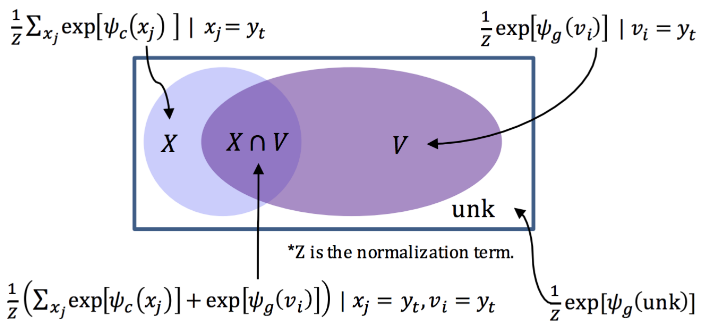

$\frac{1}{Z}\Sigma_{x_j}\exp[\psi_c(x_j)] \mid x_j = y_t$
$\frac{1}{Z}\exp[\psi_g(v_i)] \mid v_i = y_t$
X
$X \cap V$
V
unk
*Z is the normalization term.
$\frac{1}{Z}\left(\Sigma_{x_j}\exp[\psi_c(x_j)] + \exp[\psi_g(v_i)]\right) \mid x_j = y_t, v_i = y_t$
$\frac{1}{Z}\exp[\psi_g(\text{unk})]$

Figure 4.3: The illustration of the decoding probability $p(y_t|\cdot)$ as a 4-class classifier, where the white space represents the unknown words, the blue space represents the words in the source sentence, and the purple space represents the output dictionary [Gu et al., 2016]

Implementation

We built an extended *dynamic dictionary*, which contains the output dictionary plus additional words introduced by the source sentence. For each source sentence we have a *source map* that maps each source word to an index in the output dictionary if it is known, or else to an extra word. Our copy generator is an extended version of the standard generator that computes three values, the standard softmax over output dictionary; the probability of instead copying a word from the source, computed using a Bernoulli distribution; and the probability of copying a word instead that is taken directly from the attention distribution. .

Performance

When incorporating the COPYNET copying mechanism, the GLEU score has improved by 8.77 compared with the baseline (from 66.90 to 75.67), and the $M^2(F_{0.5})$ score has improved by 2.16 (from 15.71 to 17.87). The result suggests that incorporating the COPYNET copying mechanism can avoid incorrectly rectifying words that are already correct, and hence, improve the final results. The comparison of the baseline and the model incorporating the COPYNET copying mechanism is shown in Table 4.3 at the end of this chapter.

4.1.2 Grammatical Error Detection Preprocessing

Motivation

We explored using grammatical error detection as preprocessing to avoid incorrectly rectifying words that are already correct, and the details are as follows:

As mentioned in Section 4.1.1, the results of the baseline model show that the model would sometimes incorrectly rectify words that were already correct. We considered testing whether the NMT system for GEC could learn not to edit the words that are marked as correct and eventually improve the results. In detail, GED is a more straightforward task compared with GEC, both intuitively and practically. That is to say, it is possible that the correct words incorrectly rectified in the GEC task, could be labelled as correct words, and therefore might be handled correctly. Therefore, in this task, we would expect that using GED in the input sentences would improve the GEC system.

Grammatical Error Detection Tool

Ideally, a GED tagging tool would be expected to detect all grammatical errors. Meanwhile, the NMT model for the GEC is expected not to edit any of the words that are tagged as correct, and to rectify all the words that are tagged as incorrect correctly.

We used the state-of-the-art *Compositional Sequence Labelling Models for Error Detection in Learner Writing*[1] developed by Marek Rei and Helen Yan-

[1]https://github.com/marekrei/sequence-labeler

nakoudakis at the University of Cambridge [Rei and Yannakoudakis, 2016] as our GED tool. They presented the first experiments using neural network models for the task of error detection in learner writing, and proposed a novel GED framework using token-level embeddings, BiLSTMs, for context representation, and a multi-layer architecture for learning more complex features. This structure allows the model to classify each token as either being correct or incorrect, using the full sentence as context. In order to quantitatively know how accurate the GED tool [Rei and Yannakoudakis, 2016] is on our FCE test set, we compared the GED tags produced by the tool in [Rei and Yannakoudakis, 2016], with the ground-truth GED labels (see Section 2.3 for details on the dataset). We found that the precision was 0.96, and the recall was 0.97. The details of the GED model in [Rei and Yannakoudakis, 2016] tested on the FCE test dataset is shown in Table 4.1, where tag *i* is the label for *incorrect word* and tag *c* is the label for *correct word*. The results showed that the precision and recall were satisfactory, and the GED model can label words accurately.

$$precision = \frac{\sum truepositive}{\sum predictedconditionpositive} = \frac{31041}{31041 + 1225} = 0.96 \quad (4.7)$$

$$recall = \frac{\sum truepositive}{\sum conditionpositive} = \frac{31041}{31041 + 1079} = 0.97 \quad (4.8)$$

Table 4.2: Precision and Recall of the Grammatical Error Detection model tested on the FCE dataset, where tag *i* is the *incorrect* label, and tag *c* is the *correct* label.

Ground Truth	**Output Label**	**Number**	
c	c	31041	
c	i	1079	
i	c	1225	
i	i	1282	
	Precision: 0.96	**Recall:** 0.97	**Total:** 34627

Recall that the LSTM RNN is an advanced alternative to the Elman-type networks that has recently become increasingly popular (see Section 2.1.2). During the training of the model for GED tagging, the input text was lowercased, and all tokens that occurred less than twice in the training data were represented as a single UNK token. Word embeddings were set to size 300 and were initialised using the publicly released pre-trained Word2Vec vectors [Mikolov et al., 2013]. The LSTM RNNs use hidden layers of size 200 in either direction. They also added an extra hidden layer of size 50 between each of the composition functions and the output layer - this allows the network to learn a separate non-linear transformation and reduces the dimensionality of the compositional vectors. The parameters were optimised using gradient descent with an initial learning rate of 0.001, the Adam algorithm [Kingma and Ba, 2014] for dynamically adapting the learning rate and batch size of 64 sentences. $F_{0.5}$ on the development set was evaluated at each epoch, and the best model was used for the final evaluations.

An example of GED using the tool in [Rei and Yannakoudakis, 2016] is shown below, where the GED tool correctly tags each word with a GED label.

> **Example 4.1.1** ***GED tool***
>
> *Input: I* ***likes*** *eating apples.*
>
> *Hypothesis : I -c* ***likes -i*** *eating -c apples -c .-c*
>
> *where the $-c$ is the correct tag and the $-i$ is the correct tag.*

Implementation

We built two models. The first model is in an optimal state, where all the GED tags are the ground-truth; the second model is in a realistic state, where the GEC tags of the source sentences of the test set are tagged by the tool in [Rei and Yannakoudakis, 2016].

In the first task, we began by preprocessing the FCE dataset, where we labelled all source sentences with ground-truth GED tags, where each word in the source sentence is followed by its GED tag, separated by a space. In the data preprocessing step, the GEC tags are regarded as a *word in the sentence* and they are stored in the dictionary like a normal word. The NMT model allocates an initial word vector for the *-c* and *-i* tags (see Section 3.1.1 for details of the data processing step).

Then, we trained the baseline GEC system on the FCE training set. We would expect that our NMT model for GEC has learnt the correlation between the GED tags and the gold standard editions. Finally, we fed the source sentence of FCE test set and obtained the results. Subsequently, we would evaluate the hypotheses. An example of a GEC model that uses GED as preprocessing is shown as follows:

Example 4.1.2 ***A GEC Model that uses GED as preprocessing***

Input: I -c likes -i eating -c apples -c .-c

Hypothesis: I like eating apples .

where the −c is the correct tag and the −i is the correct tag.

If the results in the first task have not improved compared to the baseline, we would believe that adding GED as preprocessing would not improve the system and this attempt would be considered a failure. Otherwise, if the new GEC system is better than the baseline, we would perform the second task, which is in the realistic state.

In the second task, we would use the GEC tool in [Rei and Yannakoudakis, 2016] to tag the source sentences of the FCE test set. It is possible that, despite the first model being better than the baseline, the model using the state-of-the-art GED tool (see [Rei and Yannakoudakis, 2016]) cannot upon the final GEC results due to the poor quality of the GED tool.

Performance

In the first task, the inputs were the sentences with ground-truth grammatical errors. The GLEU score was 69.65, 2.75 higher than the baseline, and the $M^2(F_{0.5})$ score was 15.71. The results show that, ideally, the GED preprocessing can improve the GEC task. When the state-of-the-art GED tool (see [Rei and Yannakoudakis, 2016]) was used, the GLEU score result was 69.42, which is very close (GLEU score distance: 0.23) to the result of the GLEU score of an ideal condition, suggesting that the GED tool is ideal. The results also suggest that using GED as preprocessing can improve the final results. The comparison of the results is shown in Table 4.3 at the end of this chapter.

4.1.3 Part-of-speech Tagging Preprocessing

Motivation

We explored using PoS tagging as preprocessing to enhance the generalisation ability of our NMT-based GEC model for the following reasons.

PoS, like noun, verb, and adjective, is a category of words which have similar grammatical properties [Jurafsky and Martin, 2014]. Words that are assigned to the same PoS generally display similar properties concerning syntax, as they have similar properties within the grammatical structure of sentences, sometimes concerning morphology. Therefore, the information of PoS is useful in the GEC tasks.

In the GEC training, it is not possible for us to cover all grammatical errors in the training data. Hence, we want the model to generalise grammatical rules from the PoS-tagged sentence pairs. For example, the model might be able to learn that auxiliary verbs (like *might* and *can*) should be followed by an *original* verb (like *do* and *eat*). Consequently, the model would have a better *understanding* of the gold standard corrections, and might be able to correct grammatical errors that are not covered in the training set. Therefore, in this task, we would expect that tagging the PoS of the words in the input sentences would improve the GEC system.

PoS tagging

For the PoS tagging, we used the PoS tagging package of Natural Language Toolkit (NLTK) library[2] (the accuracy/precision and recall of the PoS tagger is reported to be *higher than 0.80* in the official document
[Bird and Loper, 2004]). The tag set NLTK uses is the *Universal Tagset*, described in
[McDonald et al., 2013]. An example of the NLTK PoS tagger is as follows:

> **Example 4.1.3** ***PoS tagger***
>
> *Raw: I likes eating apples .*
>
> *Processed: I -PRP likes -VBZ eating -VBG apples -NNS . -.*

The NLTK PoS tagging we used was performed using a first-order (*bigram*) hidden Markov model (HMM) (see [Collins, 2002]) tagger implemented and trained on the manually-corrected tagged Treebank-3 corpora
(see [Marcus et al., 1999]). In detail, HMMs were used to assign the correct label sequence (like GED or PoS tags) to sequential data (like text) or to assess the probability of a given label and it corresponding data sequence. The HMM models are finite state machines, which are characterised by 1) the number of states; 2) transitions between these states; and 3) output symbols emitted while in each state. The HMM is an extension to the Markov chain

[2]https://www.nltk.org/api/nltk.tag.html

[Gilks et al., 1995], where each state corresponds deterministically to a given event. In the HMM the observation follows a conditional distribution given the hidden state. HMMs share the Markov chain's assumption, being that the probability of transition from one state to another only depends on the current state, i.e., the series of states that led to the current state was not used.

Formally, an HMM can be characterised by 1) the output observation alphabet, the set of symbols which may be observed as output of the system; 2) the set of states; 3) the transition probabilities, the probability of transition to each state from a given state; 4) the output probability matrix, the probability of observing each symbol in a given state; and 5) the initial state distribution, the probability of starting in each state.

The goal of HMM decoding is to choose the PoS tag sequence that is most probable, given the observation sequence of n words w_1^n

$$\hat{t}_1^n = \arg\max_{t_1^n} P(t_1^n|w_1^n) \tag{4.9}$$

by using Bayes' rule to instead compute:

$$\hat{t}_1^n = \arg\max_{t_1^n} \frac{p(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)} \tag{4.10}$$

The optimisation problem in Equation 4.10 remains the same when dropping the denominator $P(w_1^n)$:

$$\hat{t}_1^n = \arg\max_{t_1^n} P(w_1^n|t_1^n)P(t_1^n) \tag{4.11}$$

HMM taggers make two further simplifying assumptions. The first is that the probability of a word appearing depends only on its tag and is independent of neighbouring words and tags:

$$P(w_1^n|t_1^n) \approx \Pi_{i=1}^n P(w_i|t_i) \tag{4.12}$$

The second assumption, the bigram assumption, is that the probability of a tag is dependent only on the previous tag, instead of the entire tag sequence. Note that in the NLTK PoS tagger, the bigram assumption is used.

$$P(t_1^n) \approx \Pi_{i=1}^n P(t_i|t_{i-1}) \tag{4.13}$$

Plugging the simplifying assumptions results in the following equation for the most probable tag sequence from a bigram tagger, corresponding to the emission probability and transition probability from the HMM.

$$\hat{t}_1^n = \arg\max_{t_1^n} P(t_1^n|w_1^n) \approx \arg\max_{t_1^n} \Pi_{i=1}^n \overset{emission}{P(w_i|t_i)} \overset{transition}{P(t_i|t_{i-1})} \tag{4.14}$$

Implementation

We began by preprocessing the FCE dataset. We labelled all source sentences with PoS tags generated by the NLTK PoS tagger, where each word in the source sentence is followed by its PoS tag, separated by a space. In the data preprocessing step, the GED tags are regarded as a *word in the sentence* and they are stored in the dictionary like a standard word. The NMT model allocated an initial word vector for the PoS tags (see Section 3.1.1 for details of the data processing step).

Then we trained the baseline GEC system on the FCE training set. We would expect that our NMT model for GEC has learnt the correlation between the GED tags and the gold standard editions. Finally, we fed the source sentence of the FCE test set and obtained the results. Subsequently, we would evaluate the hypotheses. An example of a GEC model that uses GED as preprocessing is shown below:

Example 4.1.4 ***GEC Model that uses PoS tagging as preprocessing***

Input: I -PRP ***likes -VBZ*** *eating -VBG apples -NNS .*

Hypothesis: I ***like*** *eating apples .*

where the PRP, VBZ, VBG, and NNS are the PoS tags.

Performance

When PoS tagging is added as preprocessing, the GLEU score improved by 6.01 compared with the baseline (from 66.90 to 72.91), and the $M^2(F_{0.5})$ score improved by 2.71 (from 15.71 to 18.42). The result suggests that using

PoS tagging as preprocessing can improve the final results. The comparison of the baseline and the model with PoS tagging preprocessing is shown in Table 4.3 at the end of this chapter.

4.1.4 GloVe Word Embedding

Motivation

We explored using GloVe word embedding to enhance the generalisation ability of our NMT-based GEC model, and the reasons are as follows:

Recall that in the baseline, the word embeddings are learned during the training. Initially, each word is assigned to a random vector, and the vectors are jointly learnt together with the baseline model. Intuitively, the rationality of the word embedding would be affected by the quality and size of the training data. In comparison, the pre-trained GloVe word embedding would contain the information of PoS tagging (words of the same PoS tag are close in space) and semantics (words of the same meaning are close in space).

While in the GEC training, it is not possible for us to cover all grammatical errors in the training data. Hence, we want the model to generalise grammatical rules from the information of PoS tags and semantics in the pre-trained word embedding. For example, if the model learns a grammatical rule of a particular word, it might be able to generalise that its synonyms should have similar behaviour in a sentence. Consequently, the model would have a better *understanding* of the gold standard corrections, and might be able to correct grammatical errors that are not covered in the training set. Therefore, the information of pre-trained word embedding might be useful in the GEC tasks, and hence, we would expect that using pre-trained GloVe ([Pennington et al., 2014]) word embedding would improve the final results.

GloVe Word Embedding

The GloVe is a global log-bilinear regression model that combines the advantages of global matrix factorisation and local context window methods (see [Pennington et al., 2014]). GloVe efficiently leverages statistical information by training only the non-zero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or individual context windows in a large corpus - Common Crawl Corpus (see [Spiegler, 2013]), and vocabulary size is $400k$. The model produces a vector space with meaningful substructure [Pennington et al., 2014] and is more reasonable than the original word embedding trained based on the FCE training data(the vocabulary size there is $1.4k$).

Implementation

Initially, each word was assigned to its GloVe vector instead of to a random vector in the baseline. Likewise, the word embeddings were updated during the training, where the vectors are jointly updated together with the model (see Section 3.1.2 for training details). In this task, the word embedding called *glove.6B.300d* (dimension: 300)[3] was used.

Performance

When GloVe word embedding is used, the GLEU score improved by 5.48 compared with the baseline (from 66.90 to 69.31), and the $M^2(F_{0.5})$ score improved by 1.07 (from 15.71 to 16.78). The result suggests that using GloVe word embedding can improve the final results. The comparison of the baseline and the model using GloVe word embedding is shown in Table 4.3.

4.2 Combining Improvements

In Section 4.1, we found that all of the possible improvements we proposed showed a substantial performance gain. Next, we explored whether combining the individual improvements can further improve the model.

4.2.1 Combining Preprocessings

Motivation

In Section 4.1, we found that both GED and PoS preprocessing would improve the final results. Considering both GED and PoS are important information in GEC we explored combining GED and PoS preprocessing. We would expect that the model could use the information of both the GED

[3]https://nlp.stanford.edu/projects/glove/

and the PoS, and learn GEC from the labelled training set. However, it is also possible that the GEC network would fail to handle the combined information.

Example 4.2.1 ***Model that uses combined preprocessing***

Input: I -PRP-c ***likes -VBZ-i*** *eating -VBG-c apples -NNS-c .*

Hypothesis: I ***like*** *eating apples .*

Where the PRP, VBZ, VBG, and NNS are the PoS tags, and the $-c$ is the correct tag and the $-i$ is the correct tag.

Performance

When both GED and PoS tagging are added as preprocessing, the GLEU score improved by compared with the baseline (from 66.90 to 73.72), and the $M^2(F_{0.5})$ score improved by (from 15.71 to 28.72). The result suggests that using both GED and PoS tagging as preprocessing is the current best way to improve the final results. The comparison of the baseline and the model with both GED and PoS tagging preprocessing is shown in Table 4.3 at the end of this chapter.

4.2.2 Applying COPYNET to Other Improvements

Motivation

In order to obtain a better performance, we would want to correct the grammatical errors and keep the correct words. From Section 4.1, we can see that copy attention mechanism can improve the GLEU score, as it successfully alleviated incorrectly-rectified words that were already correct, and preprocessing (GED and PoS tagging) can improve the $M^2(F_{0.5})$ score as they help to correct grammatical errors (see Table 4.3). Since both features are desired, we considered applying the copy attention mechanism to the model that uses GED and PoS tagging preprocessing. Similarly, we also tested whether copy attention can improve the model that uses GloVe word embedding. As explained in 4.2.1, we are not combining copy attention mechanism, preprocessing and GloVe word embedding.

Performance

The results and comparisons are shown in Table 4.3 at the end of this chapter. Notably, this is the current best model of our thesis. The GLEU score of this model is 81.29, and the $M^2(F_{0.5})$ score is 34.37, of which the GLEU score, to the best knowledge of the author, outperforms the state-of-the-art result of [Yannakoudakis et al., 2017] (71.76) on the FCE test set.

4.3 Examples of the Hypotheses

We manually read through the hypotheses of the best model (baseline + COPYNET + Rei GED + PoS, GLEU score: 81.29, $M^2(F_{0.5})$ score: 34.37) and selected some typical failed cases and successful cases to intuitively have a better understanding of what kinds of grammatical errors the best model can correct and what they cannot. For some of the failure types, possible solutions are also proposed.

4.3.1 Failure Analysis

In this section, we show some typical failure cases for analysis. The failure types include: 1) failure in spotting grammatical errors; 2) failed as it incorrectly edits correct words; 3) successful in spotting an error but failed to correct it; 4) successful in replacing the incorrect words with grammatically correct words but with the meanings changed; and 5) trapped in an infinite loop. The details are as follows:

Type: Failure in spotting grammatical errors

In some cases, the model failed to spot grammatical errors, and the hypotheses still had the same grammatical errors as the source sentences.

Example 4.3.1 ***Failed in spotting grammatical errors***

Input: There are plenty of ***churchs*** *, cathedrals , and* ***olds cotages*** *.*

Hypothesis: There are plenty of ***churchs*** *, cathedrals , and* ***olds cotages*** *.*

Ground Truth: There are plenty of ***churches*** *, cathedrals , and* ***old cottages*** *.*

From the example, we can see that in the input sentence, there are grammatical errors, like the word **churchs** should be **churches** as this plural noun is ends in **ch**. However, the hypothesis copied the incorrect words and did not correct them. One possible reason for failing to spot grammatical errors is that the model failed to learn this kind of error, or the particular error was not included in the training set, and the model failed to generalise this kind of error from other training examples.

Using more training data would help in learning more GEC and improve the final results. Also, character-level NMT GEC might help in generalising spelling mistakes [Ji et al., 2017]. For example, in word-level GEC, when a word ends in ***ch***, then ***es*** instead of ***s*** should be appended at the end. The input sentence in the training set is *I saw some* ***churchs*** and the hypothesis is *I saw some churches.* The model would successfully learn to correct the incorrect word ***churchs*** to ***churches***. However, it cannot generalise this study case to know that the word ***branchs*** should be ***branches***. In comparison, character-level NMT GEC is able to deduce that the letters ***ch*** should be followed by ***es*** instead of ***s***. Additionally, if we can improve the GED of state-of-the-art, the final results would be improved as well (see Table 4.3 at the end of this chapter where the model that uses ground-truth GED is better than the model that uses the model in [Rei and Yannakoudakis, 2016]).

Type: Failed as it incorrectly edits correct words

In some cases, words that were already correct in the source sentences were incorrectly edited. When this happens, the meanings of sentences are often changed and sometimes would even introduce grammatical errors that are not in the source sentence. An example is shown below:

> **Example 4.3.2** ***Failed as it incorrectly edits correct words***
>
> *Input: The* ***Hotel*** *is near the station and is call " **Grand** Placad " .*
>
> *Hypothesis: The* ***saleswomen*** *is near the station and is call " **organise** Placad " .*
>
> *Ground Truth: The* ***Hotel*** *is near the station and is call " **Grand** Place" .*

In this example, we can see that the words **Hotel** and **Grand** are correct proper nouns; however, the model incorrectly edited these words which changed the meaning of the words and also introduced new grammatical errors. Empirically, unseen proper nouns are especially prone to being incorrectly edited.

In Section 4, we found that applying copy attention mechanism is the best improvement in avoiding touching correct words. Using different hyperparameters or developing other improvements in copy attention mechanism might improve the model in copying correct words.

Type: Successful in spotting an error but failed to correct it

In some cases, the model spotted that some words were incorrect, but the model failed to correct them. The hypotheses still have grammatical errors at the same positions. An example is as follows: the model spotted that the word ***luxuery*** is incorrect, but replaced it with ***most***, which does not make sense, nor it is grammatically correct. The possible solution for these kinds of errors is the same as the solution in Section 4.3.1.

> **Example 4.3.3** ***Successful in spotting the error, but failed to correct it***
>
> *Input: In addition to this , we choose the* **luxuery** *Palace Hotel , which is comfortable enough and in a good location .*
>
> *Hypothesis: In addition to this , we chose the* ***most*** *Palace Hotel , which is comfortable enough and in a good location .*
>
> *Ground Truth: In addition to this , we have chosen the* ***luxury*** *Palace Hotel , which is comfortable enough and in a good location .*

Type: Successful in replacing the incorrect word with a grammatically correct word, but the meaning is changed

In some cases, the model learned some grammatical rules and successfully spotted some grammatical errors in the input sentences. The model replaced the incorrect word with a grammatically correct word, although not the intended word, consequently changing the meaning of the sentence. Usually, the predicted new words are the synonyms of the ground-truth words. The possible solution for these kinds of errors is the same as the solution in Section 4.3.1. An example is as follows:

> **Example 4.3.4** ***Successfully replaced the incorrect words with a grammatically correct word, but the meaning was changed***

Input: best ***regardes***

Hypothesis: best ***wishes***

Ground Truth: best ***regards***

In this case, the model spotted that the word ***regardes*** is incorrect and replaced it with wishes. We checked the training set and found that the phrase ***best wishes*** was included while ***best regards*** was not, which can explain why the model predicted ***best wishes*** instead of ***best regards***. Grammatically ***best wishes*** is correct, but the meaning is changed.

Type: Trapped in an infinite loop

When handling long sentences (empirically longer than 100 words), the word prediction would be trapped in an infinite loop from some point (empirically, the frequency is 1/2000). In these cases, we can observe that a few words are looping in the hypothesis. An example is as follows:

Example 4.3.5 ***Example of being trapped in an infinite loop***

Input: However I feel that at the moment there should emerge more grups to control this development because the industrial polution that ***it has been created and it seems that its very little what has been done at the moment in relation to our embiroment .***

Hypothesis: However I feel that at the moment there should spend more necessary to control this development because the industrial pollution that ***it has been created and it seems that it seems***

Ground Truth: However , I feel that at the moment there should emerge more groups to control this development because of the industrial pollution that ***has been created and it seems that it 's very little , what is being done at the moment in relation to our environment .***

From the example, we can see that after phrase ***has been created and*** , the model repeatedly predicted ***it seems that*** until the end of the output sentence. Linguistically, the phrase *it seems that* is most likely to be followed by *it*; and *it* is most likely to be followed by *seems that*; and hence a prediction loop is formed.

For this kind of error, we can force the NMT model to block the repetition of n-grams in translation. Empirically, n can be set to a number between two and five[4] (see [Klein et al., 2017]).

4.3.2 Successful Examples

We also picked some cases where the GECs were successful, or the input sentences were correct, and the model did not edit any words.

Type: Successful in GEC

Example 4.3.6 ***Successful in GEC***

Input: ***Your*** *sincerely, Maria Smith*

Hypothesis: ***Yours*** *sincerely, Maria Smith*

Ground Truth: ***Yours*** *sincerely, Maria Smith*

Type: Successful in not editing sentences that are already correct

Example 4.3.7 ***Successful cases in not touching sentences that correct***

Input: Being sincere , most of us would go to the telephone .

Hypothesis : Being sincere , most of us would go to the telephone .

Ground Truth: Being sincere , most of us would go to the telephone .

4.4 Results Summary

From the $M^2(F_{0.5})$ score column in Table 4.3, we can see that, in general, all models can successfully correct some grammatical errors, suggesting that the model learned GEC rules from the training. However, from the GLEU scores, we can see that only three out of the nine models performed better than the original sentence, i.e., in general, the hypotheses of most of the

[4]OpenNMT-py has this feature, see http://opennmt.net/OpenNMT-py/options/translate.html

models were worse than the source, as the GEC models would sometimes incorrectly rectify words that were already correct, which can be solved when copy attention is used.

As expected, all individual improvements can improve the baseline, as both the $M^2(F_{0.5})$ score and the GLEU score were higher. Considering fluency, applying copy attention is the best individual improvement.

According to the ranking of the $M^2(F_{0.5})$ score, we can see that the best individual improvement to spot and correct errors is to add GED as preprocessing; while the least useful individual improvement is using GloVe word embedding. Linguistically, this result makes sense. The model that is trained with GED has a better understanding of how to correct words, and the model with copy attention is trained not to touch the correct words.

According to the ranking of the GLEU score, we can see that the individual improvement that is best in fluency, readability and not touching correct words in the original sentence is adding copy attention mechanism; while the least useful individual improvement is using GloVe word embedding. Linguistically, this result also makes sense. The model that is trained with copy attention mechanism will copy the words that are probably correct and avoid incorrectly editing correct words. In comparison, the model that used GloVe word embedding would sometimes replace the correct words in the original sentences with synonyms, as their word vectors are close, introducing new grammatical errors and lowering the GLEU score.

Notably, GloVe word embedding can improve both the GLEU score and the $M^2(F_{0.5})$ score but is the least significant improvement. The meaning of the words would provide useful information but is not as important as the GED tags or PoS tags.

When we combine the four individual improvements, all of the models are better than the individual improvements. Notably, applying copy attention with PoS tagging and GED preprocessing is the best model in both the GLEU score and the $M^2(F_{0.5})$ score (GLEU: 81.29, $M^2(F_{0.5})$ score: 34.3718).

In Table 4.3, the symbol + means *adding* or *using/applying*; **ground-truth GED** is the GEC preprocessing in its optimal state that uses true GED tags; **Rei GED** is the GED tool in [Rei and Yannakoudakis, 2016] that would contain some wrong tags; **GloVe** is the GloVe word embedding; **PoS** is the PoS tagging preprocessing; and **copy attention** is the COPYNET attention copying mechanism.

Table 4.3: Testing Results

Model	**GLEU**	$M^2(F_{0.5})$	**Distance to Baseline (GLEU)**	**Distance to Baseline** ($M^2(F_{0.5})$)	**Ranking (GLEU)**	**Ranking** ($M^2(F_{0.5})$)
source	74.79	0	N/A	N/A	N/A	N/A
baseline	66.90	15.71	N/A	N/A	N/A	N/A
baseline + GloVe	69.31	16.78	5.48	1.07	6	8
baseline + ground-truth GED	69.65	**19.79**	2.75	**4.07**	7	3
baseline + Rei GED	69.42	19.26	2.52	3.54	8	4
baseline + PoS	72.91	18.42	6.01	2.71	5	5
baseline + copy attention	**75.67**	17.87	**8.77**	2.16	3	6
baseline + PoS + Rei GED	73.72	28.72	6.82	13.01	4	2
baseline + copy attention + GloVe	76.16	17.06	9.26	1.35	2	7
baseline + copy attention + PoS + Rei GED	**81.29**	**34.37**	**18.66**	**14.39**	1	1

Chapter 5

Conclusions and Future Research

5.1 Conclusions

In this thesis, we studied NMT for GEC and discussed the extensions of NMT GEC in English text. We treated GEC as a translation task from ungrammatical to grammatical English. We presented a survey of different approaches to improve the baseline GEC model.

We implemented a baseline word-level NMT GEC model based on the model in [Yuan and Briscoe, 2016] using the OpenNMT framework. We trained our baseline model using the FCE dataset. We have shown that our baseline GEC models can correct some grammatical errors ($M^2(F_{0.5})$ score: 15.71) but is not as good as the model in [Yuan and Briscoe, 2016] ($M^2(F_{0.5})$ score: 53.49, GLEU score: 71.16) and finds difficulty in not editing words that are correct, resulting in a relatively low GLEU score (66.90).

We experimented on four individual improvements, including applying copy attention mechanism, using the GloVe word embedding, adding GED preprocessing and adding PoS tagging preprocessing. In addition, we explored combining the individual improvements. The GLEU score of our best model (using GED preprocessing, PoS preprocessing and COPYNET mechanism) was 81.29, and the $M^2(F_{0.5})$ score was 34.37, of which the GLEU score, to the best knowledge of the author, outperformed the state-of-the-art result of [Yannakoudakis et al., 2017] (71.76) on the FCE test set.

The main contributions of this thesis are four extensions to the baseline, including 1) applying the copy attention mechanism; 2) using Global Vectors for Word Representation (GloVe) word embedding; 3) adding grammatical error detection (GED) preprocessing; and 4) adding part-of-speech (PoS) tagging preprocessing. These improvements show a substantial performance gain. We then further explored these individual improvements in combination, which resulted in a model with a state-of-the-art GLEU score performance on the FCE test dataset. Finally, we reported the results of all of our models on the well-known publicly available FCE test set, which can be used for future cross-system comparisons by other researchers.

In more detail, our best model was developed based on the baseline model with the addition of a copy mechanism, PoS tagging preprocessing and GED preprocessing. The GLEU score of the best model was 81.29, and the $M^2(F_{0.5})$ score was 34.37, meaning that the GLEU score, to the best knowledge of the author, outperformed the state-of-the-art result of [Yannakoudakis et al., 2017] (71.76) on the FCE test set.

In our work, the object language for GEC is English. However, our research methodology could be easily adapted and applied to other languages. Equivalent results should be expected.

5.2 Future Research

In our future work, we will develop our NMT models to improve the performance of the GEC system in correcting complex errors. For future research, we propose a few ideas which can improve the quality of the GEC system.

5.2.1 Character-Level GEC

Character-level NMT GEC might help in learning general spelling mistakes [Ji et al., 2017], and hence correct the mis-spelt OOV words that are not in the training set. For example, when a word ends in ***ch***, then ***es*** instead of ***s*** should be appended at the end.

5.2.2 Multiple References

For some errors, there is more than one possible correction. For example, *The **child are** playing.* The correction might be *The **child is** playing.* or *The*

children are *playing.* according to the context. Therefore, a dataset with multiple references is suggested. Notably, both $M_2(F_{0.5})$ score and GLEU score calculations support multiple references.

5.2.3 Training on Long-sentence GEC

In GEC, most of the errors are interacting errors and may depend on long-range contextual information. MT models find difficulty in learning the global contextual information for correcting these errors. We manually read through the hypotheses of the models and found that they performed relatively poorly in long-sentence GEC. Additionally, we found that there are very few long sentences (more than 100 words) in the training set (less than 1%). Hence, we suggest putting more long sentences in the training set in order to improve the model.

Bibliography

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Bird and Loper, 2004] Bird, S. and Loper, E. (2004). Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.

[Breene, 2016] Breene, K. (2016). Which countries are best at english as a second language?

[Brockett et al., 2006] Brockett, C., Dolan, W. B., and Gamon, M. (2006). Correcting esl errors using phrasal smt techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 249–256. Association for Computational Linguistics.

[Bustamante and León, 1996] Bustamante, F. R. and León, F. S. (1996). Gramcheck: A grammar and style checker. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 175–181. Association for Computational Linguistics.

[Casas et al., 2018] Casas, N., Fonollosa, J. A., and Costa-jussà, M. R. (2018). A differentiable bleu loss. analysis and first results. *Departa-*

ment de Teoria del Senyal i Comunicacions - Ponencies/Comunicacions de congressos.

[Chen et al., 2018] Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Parmar, N., Schuster, M., Chen, Z., et al. (2018). The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849.*

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078.*

[Collar, 2016] Collar, C. (2016). Understanding lstm networks. `http:\/\/colah.github.io\/posts\/2015-08-Understanding\-LSTMs/`. Accessed: 2018-04-30.

[Collar, 2017] Collar, C. (2017). Why lstms stop your gradients from vanishing: A view from the backwards pass. `https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html`. Accessed: 2018-04-30.

[Collins, 2002] Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

[Crystal, 2004] Crystal, D. (2004). *The Cambridge encyclopedia of the English language.* Cambridge University Press, 2 edition.

[Dahlmeier and Ng, 2012] Dahlmeier, D. and Ng, H. T. (2012). Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572. Association for Computational Linguistics.

[Deloche, 2017] Deloche, F. (2017). File:recurrent neural network unfold.svg — wikimedia commons, the free media repository. [Online; accessed 4-June-2018].

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[Felice et al., 2014] Felice, M., Yuan, Z., Andersen, Ø. E., Yannakoudakis, H., and Kochmar, E. (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.

[Gal and Ghahramani, 2016] Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

[Gilks et al., 1995] Gilks, W. R., Richardson, S., and Spiegelhalter, D. (1995). *Markov chain Monte Carlo in practice*. CRC press.

[Gojenola and Oronoz, 2000] Gojenola, K. and Oronoz, M. (2000). Corpus-based syntactic error detection using syntactic patterns. In *Proceedings of the workshop on Student research*, pages 24–29. Morgan Kaufmann Publishers Inc.

[Graves et al., 2013] Graves, A., Jaitly, N., and Mohamed, A.-r. (2013). Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE.

[Gu et al., 2017] Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. (2017). Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

[Gu et al., 2016] Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

[Han et al., 2004] Han, N.-R., Chodorow, M., and Leacock, C. (2004). Detecting errors in english article usage with a maximum entropy classifier trained on a large, diverse corpus. In *LREC*.

[Heidorn et al., 1982] Heidorn, G. E., Jensen, K., Miller, L. A., Byrd, R. J., and Chodorow, M. S. (1982). The epistle text-critiquing system. *IBM Systems Journal*, 21(3):305–326.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Ji et al., 2017] Ji, J., Wang, Q., Toutanova, K., Gong, Y., Truong, S., and Gao, J. (2017). A nested attention neural hybrid model for grammatical error correction. *arXiv preprint arXiv:1707.02026*.

[Junczys-Dowmunt and Grundkiewicz, 2014] Junczys-Dowmunt, M. and Grundkiewicz, R. (2014). The amu system in the conll-2014 shared task:

Grammatical error correction by data-intensive and feature-rich statistical machine translation. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 25–33.

[Jurafsky and Martin, 2014] Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing*, volume 3. Pearson London:.

[Kao et al., 2013] Kao, T.-H., Chang, Y.-W., Chiu, H.-W., Yen, T.-H., Boisson, J., Wu, J.-C., and Chang, J. S. (2013). Conll-2013 shared task: Grammatical error correction nthu system description. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 20–25.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Klein et al., 2017] Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

[Koehn, 2009] Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.

[Kukich, 1992] Kukich, K. (1992). Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)*, 24(4):377–439.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer.

[Lewis et al., 2004] Lewis, J. R., Ballard, B. E., Hanson, G. R., Ortega, K. A., Vanbuskirk, R. E., and Keller, A. (2004). Method and system for proofreading and correcting dictated text. US Patent 6,760,700.

[Luong et al., 2017] Luong, M., Brevdo, E., and Zhao, R. (2017). Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*.

[Luong et al., 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

[Marcus et al., 1999] Marcus, M., Santorini, B., Marcinkiewicz, M. A., and Taylor, A. (1999). Treebank-3 ldc99t42. *CD-ROM. Philadelphia, Penn.: Linguistic Data Consortium*.

[McDonald et al., 2013] McDonald, R., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., Hall, K., Petrov, S., Zhang, H., Täckström, O., et al. (2013). Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 92–97.

[Meng et al., 2015] Meng, F., Lu, Z., Wang, M., Li, H., Jiang, W., and Liu, Q. (2015). Encoding source language with convolutional neural network for machine translation. *arXiv preprint arXiv:1503.01838*.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mutton et al., 2007] Mutton, A., Dras, M., Wan, S., and Dale, R. (2007). Gleu: Automatic evaluation of sentence-level fluency. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 344–351.

[Napoles and Callison-Burch, 2017] Napoles, C. and Callison-Burch, C. (2017). Systematically adapting machine translation for grammatical error correction. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 345–356.

[Napoles et al., 2015] Napoles, C., Sakaguchi, K., Post, M., and Tetreault, J. (2015). Ground truth for grammatical error correction metrics. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 588–593.

[Ng et al., 2014] 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. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14.

[Nicholls, 2003] Nicholls, D. (2003). The cambridge learner corpus: Error coding and analysis for lexicography and elt. In *Proceedings of the Corpus Linguistics 2003 conference*, volume 16, pages 572–581.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

[Park et al., 1997] Park, J. C., Palmer, M. S., and Washburn, C. (1997). An english grammar checker as a writing aid for students of english as a second language. In *ANLP*, page 24.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Platt et al., 1999] Platt, J. et al. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

[Rei and Yannakoudakis, 2016] Rei, M. and Yannakoudakis, H. (2016). Compositional sequence labeling models for error detection in learner writing. *arXiv preprint arXiv:1607.06153*.

[Rozovskaya and Roth, 2011] Rozovskaya, A. and Roth, D. (2011). Algorithm selection and model adaptation for esl correction tasks. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 924–933. Association for Computational Linguistics.

[Sak et al., 2014] Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*.

[Spiegler, 2013] Spiegler, S. (2013). Statistcs of the common crawl corpus 2012. Technical report, Technical report, SwiftKey.

[Srivastava et al., 2015] Srivastava, N., Mansimov, E., and Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Tetreault and Chodorow, 2008] Tetreault, J. R. and Chodorow, M. (2008). The ups and downs of preposition error detection in esl writing. In *Proceed-*

ings of the 22nd International Conference on Computational Linguistics-Volume 1, pages 865–872. Association for Computational Linguistics.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[Van Weijen, 2012] Van Weijen, D. (2012). The language of (future) scientific communication. *Research trends*, 31(November).

[Yannakoudakis et al., 2011] Yannakoudakis, H., Briscoe, T., and Medlock, B. (2011). A new dataset and method for automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 180–189. Association for Computational Linguistics.

[Yannakoudakis et al., 2017] Yannakoudakis, H., Rei, M., Andersen, Ø. E., and Yuan, Z. (2017). Neural sequence-labelling models for grammatical error correction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2795–2806.

[Yuan, 2017] Yuan, Z. (2017). Grammatical error correction in non-native English. Technical Report UCAM-CL-TR-904, University of Cambridge, Computer Laboratory.

[Yuan and Briscoe, 2016] Yuan, Z. and Briscoe, T. (2016). Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386.

[Zhang and Zong, 2016] Zhang, J. and Zong, C. (2016). Exploiting source-side monolingual data in neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545.

Example 5.2.1

Baseline input: I likes eating ***apples*** *.*

Baseline hypothesis: I like eating ***oranges*** *.*

New input: I -c likes -i eating -c ***apples -c*** *. -c*

New hypothesis: I like eating ***apples*** *.*

-i: incorrect

-c: correct