Jiaxin Wang

A Recursive Recurrent Neural Network Decoder for Grammatical Error Correction

Computer Science Tripos – Part II

Emmanuel College

April 19, 2022

Proforma

Candidate Number:	-
Project Title:	A Recursive Recurrent Neural Network Decoder for Grammatical Error Correction
Examination:	Computer Science Tripos – Part II, 2022
Word Count:	2894 [1]
Code line count:	1127
Project Originator:	-
Supervisor:	Dr Zheng Yuan, Dr Christopher Bryant

Original Aims of the Project

-

Work Completed

-

Special Difficulties

-

[1] This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

Declaration

I, Jiaxin Wang of Emmanuel College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date [date]

Contents

List of Figures

Acknowledgements

-

Todo list

Chapter 1

Introduction

This project concerns my implementation of a *recursive recurrent neural network* model (R^2NN) proposed by Liu et al. (2014) [6] This is to be integrated in a statistical machine translation (SMT) system to attempt the task of grammatical error correction.

More details

1.1 Motivation

Grammatical Error Correction (GEC) is the task of producing a grammatically correct sentence given a potentially erroneous text while preserving its meaning. The main motivation behind this task lies in its role in helping learners of a foreign language understand the language better. In addition, native speakers can make use of such GEC tools to avoid mistakes in their writing. Being an ESL (English as a second language) learner myself, I would like to investigate models that can be used for the task of GEC in English, and I hope that it can benefit people who may find GEC tools useful.

1.2 Problem Overview

The task of Grammatical Error Correction can be seen as a machine translation process, where the input is a text which may contain grammatical errors, and the output is an error-free text. This project uses a statistical machine translation (SMT) approach to solve GEC.

A recursive recurrent neural network (R^2NN) was proposed by Liu et al. (2014) [6] for SMT. The goal of this project is to implement the proposed model to be used for GEC. The model should aim to correct all types of errors, namely grammatical, lexical, and orthographical errors. Its performance will be evaluated against a baseline SMT system, Moses[5].

1.3 Related Work

-

Chapter 2

Preparation

2.1 Starting Point

This project is based on the idea presented in the paper *A Recursive Recurrent Neural Network for Statistical Machine Translation* [6]. An R^{2}NN model was proposed, but the implementation details are not given in the paper.

The Part IB Computer Science Tripos course Artificial Intelligence[1] gives an introduction to neural networks and explains how forwarding and backpropagation works. Prior to this project I did not have any coding experience with neural networks. I have found it useful to follow the coding examples on PyTorch[2] website.

An example SMT system is available on Moses[3] website. It provides a detailed tutorial on how to install Moses, how to prepare corpus and how to train the SMT.

2.2 Theory

2.2.1 SMT system

A typical SMT system consists of four main components: The language model (LM), the translation model (TM), the reordering model and the decoder[7]. The LM computes the probability of a given sequence being valid. The TM builds a translation table which contains mappings of words/phrases between source and target corpora. The reordering model learns about phrase reordering of translation. The decoder finds a translation candidate who is most likely to be the translation of the source sentence. In the task of GEC, this would be the most probable correction to the original erroneous sentence.

Level of detail?

[1] https://www.cl.cam.ac.uk/teaching/2021/ArtInt/
[2] https://pytorch.org/tutorials/
[3] https://www.statmt.org/moses/?n=Moses.Baseline

2.2.2 Moses SMT

The Moses baseline system uses KenLM as the language model. For the translation model, it uses GIZA++ to obtain a word alignment model. The model should be trained to produce a phrase table and associated scores. Reordering tables are also created during this process. Eventually, the Moses decoder will find the best translation candidate given input based on the scores it calculated in previous stages.

2.2.3 Feedforward Neural Network

A feedforward neural network consists of three parts: an input layer, one or more hidden layers, and an output layer. In a feedforward neural network, data only flows in one direction (forward) from input to output. Figure 2.1 shows an example of a feedforward neural network with one hidden layer.

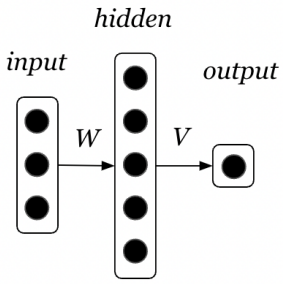

Figure 2.1: Example of a feedforward neural network

2.2.4 Recurrent Neural Network (RNN)

Recurrent neural networks are usually used to deal with sequences. They allow access to the input data as well as past data to compute the next state. As shown in Figure 2.2, the hidden layer h_t is computed using both input x_t at time t and the hidden state h_{t-1} which contains the history information from time 0 to $t-1$. For each timestep t, the hidden state can be expressed as:

$$h_t = Wx_t + Uh_{t-1}$$

2.2.5 Recursive Neural Network (RvNN)

A recursive neural network has a tree-like structure. Figure 2.3 illustrates an example of a basic RvNN architecture. The parent node representation is computed from its child nodes' representation as follows:

$$p_{n,n+1} = f(W[x_n; x_{n+1}])$$

where f is the activation function. The same weight matrix W will be applied recursively over the input.

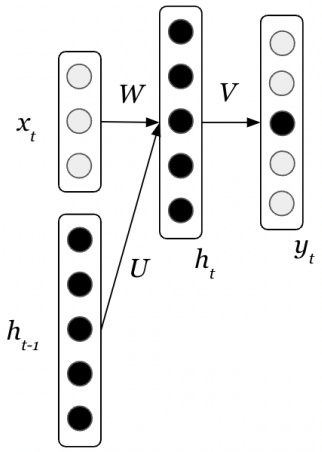

Figure 2.2: Example of a recurrent neural network (RNN)

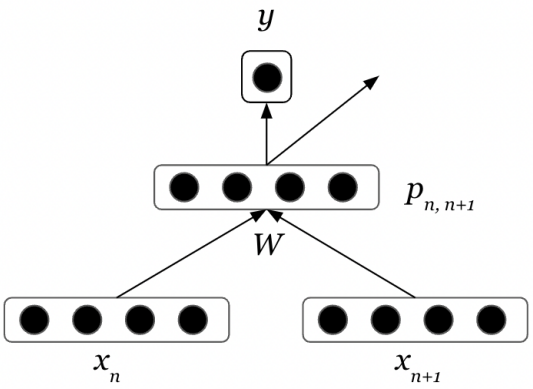

Figure 2.3: Example of a recursive neural network (RvNN)

2.2.6 Recursive Recurrent Neural Network (R^2NN)

The R^2NN proposed by Liu et al.[6] combines the features of RvNN and RNN. It has a tree-like structure similar to RvNN, with recurrent vectors added to integrate global information. As shown in Figure 2.4, $s^{[l,m]}$ and $s^{[m,n]}$ is the representation of child nodes $[l, m]$ and $[m, n]$. The recurrent input vectors, $x^{[l,m]}$ and $x^{[m,n]}$ are added to the two child nodes respectively. They encode the global information, such as language model scores and distortion model scores. A third recurrent input vector $x^{[l,n]}$ is added to the parent node $[l, n]$. The parent node representation is computed as

$$s_j^{[l,n]} = f(\sum_i \hat{x}_i^{[l,n]} w_{ji})$$

where $\hat{x}$ is the concatenation of vectors $[x^{[l,m]}; s^{[l,m]}; x^{[m,n]}; s^{[m,n]}]$, and f is the $HTanh$ function. The output, $y^{[l,n]}$, is computed as

$$y^{[l,n]} = \sum_j ([s^{[l,n]}; x^{[l,n]}])_j v_j$$

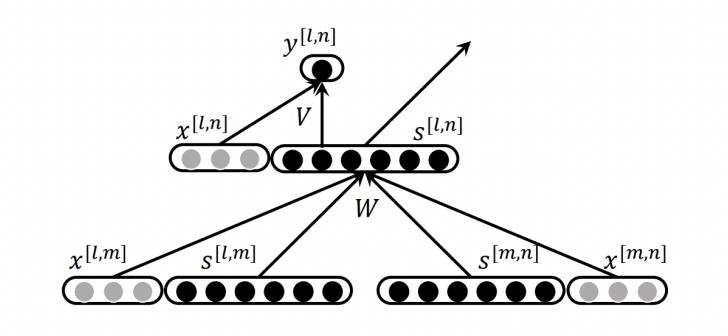

Figure 2.4: Recursive recurrent neural network (Liu et al., 2014, p.1494)

2.3 Requirement Analysis

Based on the Project Structure section from my project proposal, the following requirements have been identified:

Data preprocessing

- Data should be prepared in a form that is accepted by Moses SMT and R^{2}NN SMT
- Preprocessing should be done carefully to avoid accidentally correcting some of the grammatical errors
  - E.g. capitalisation errors may go undetected if all sentences are lowercased during data preprocessing

Moses SMT for GEC

- A language model should be trained to model the probability of a given sentence being valid
- A translation model should be trained to construct a phrase translation table
- A reordering model should be trained to learn the reordering of phrases
- With the above three models and Moses decoder, a complete Moses SMT system should be built

R^{2}NN SMT for GEC

Following the R^{2}NN paper by Liu et al.[6],

- Phrase pair embeddings (PPE) should be learned by building a one-hidden-layer neural network and a recurrent neural network
- A recursive recurrent neural network (R^{2}NN) should be built and used as a decoder

Evaluation

- The performance of both SMT systems should be evaluated using F0.5 scores

2.4 Choice of Tools

2.4.1 Programming Language

Python is chosen to be the main programming language as it provides many libraries that are commonly used for natural language processing. For this project I will be using `python 3.8` and PyCharm as my IDE.

2.4.2 Libraries

PyTorch
The `PyTorch`[4] library is one of the most popular machine learning frameworks. There are other similar libraries (such as `TensorFlow`) but I find `PyTorch` tutorials are easier to follow.

NumPy
My project is likely to involve statistical processing. I would be using the `NumPy`[5] library for this purpose.

pandas
`pandas`[6] is a powerful library for processing tabular data. This would be used to manipulate phrase tables in my project.

2.4.3 Dataset

The dataset introduced in BEA 2019 Shared Task[1] will be used in this project. I chose the corpora (**FCE v2.1**) which is immediately downloadable from the website[7] to start with. The corpora have been standardised to be easily evaluated by ERRANT[2, 4]. ERRANT is a toolkit used to annotate parallel data and compare hypothesis against reference to produce various evaluation metrics including F0.5 score. I may request other corpora as an extension of my project.

For language model training, I will be using the **One Billion Word** dataset[3]. This is a dataset used for language modeling and is available on GitHub[8].

2.4.4 Version Control and Backup

Git will be used for version control. The entire project, including all the written code and my dissertation, will be pushed to GitHub regularly.

[4]https://pytorch.org/
[5]https://numpy.org/
[6]https://pandas.pydata.org/
[7]https://www.cl.cam.ac.uk/research/nl/bea2019st/#data
[8]https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark

Chapter 3

Implementation

This chapter describes the implementation of a Moses baseline SMT system and a R^2NN SMT system. Since the main purpose of this project is to compare the performance of our R^2NN decoder against Moses decoder, the R^2NN system should use the same language model, translation model and reordering model as Moses. For the translation model, the R^2NN paper[6] proposed a *translation confidence based phrase pair embedding* (TCBPPE) to be used with the R^2NN decoder. The TCBPPE will be based on a phrase table which is produced by Moses translation model. In the end, the R^2NN decoder will make use of language model scores, translation model scores, reordering model scores from Moses as well as TCBPPE to find the best translation candidate.

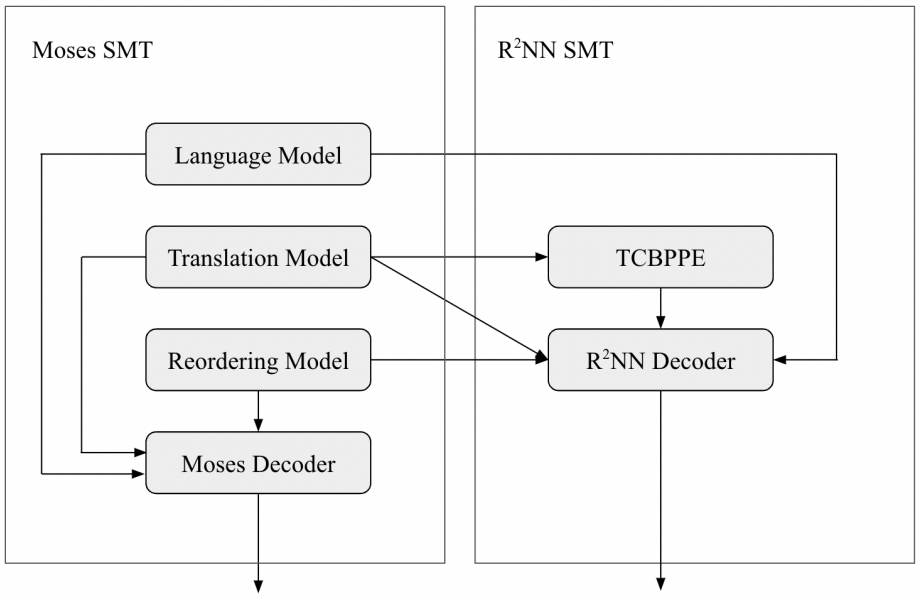

Figure 3.1: Overview of Moses SMT and R^2NN SMT

3.1 Repository Overview

-

3.2 Setup

FCE Dataset

The FCE dataset was provided in m2 format. However, Moses requires parallel data which is aligned at the sentence level. An example taken from the m2 file looks like this:

```
S Her friend Pat had explained the whole story at her husband .
A 8 9|||R:PREP|||to|||REQUIRED|||-NONE-|||0
```

[This means that for source sentence S, the word at position 8 to 9 (`at`) should be corrected to `to`.]

The m2 format needs to be converted to sentence-aligned data to be used by Moses. Two files are generated from this, where the source file contains
`Her friend Pat had explained the whole story at her husband .`

and the target file contains
`Her friend Pat had explained the whole story to her husband .`

One Billion Word Dataset

I downloaded the One Billion Word dataset for language model training. However, the files were too large to be handled by my machine. Since the dataset consists of fifty files, I decided to randomly choose 10 files from them and concatenated these files to be used in language model training.

3.3 Moses Baseline

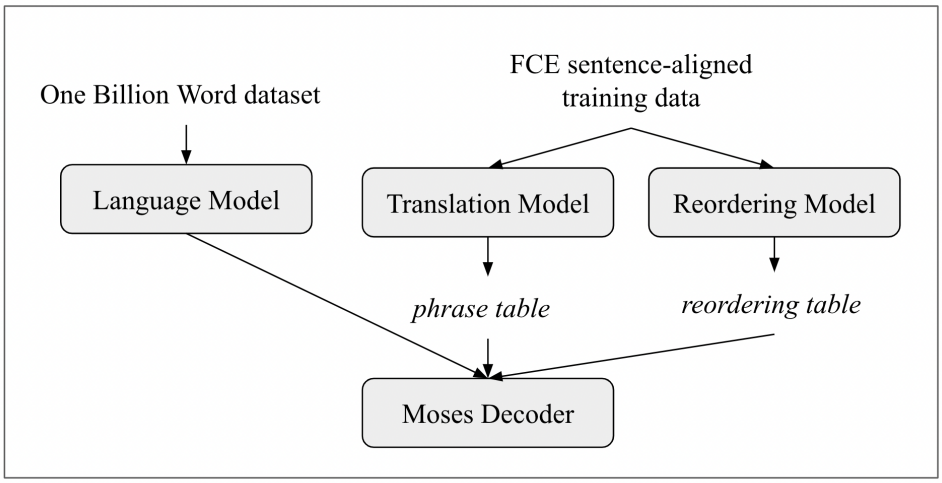

Figure 3.2: Overview of Moses SMT

3.3.1 Language Model

The One Billion Word dataset is used for language model training. In the R²NN paper, a 5-gram language model is used. Here, I used KenLM which came with Moses installation to train a 5-gram language model. An example query of the trained language model looks like this:

Example query of LM

3.3.2 Translation Model and Reordering Model

The translation model is trained with FCE sentence-aligned data. There are three components in translation model training: word alignment, phrase extraction and scoring. Word alignment is obtained using GIZA++, a toolkit to train word alignment models. Phrases are then extracted from our training files and a phrase table is produced. The phrase table contains phrase translation pairs and the scores associated with each pair.

source	target	scores	alignment	counts
As we	As we	1 0.901657 0.846154 0.90511	0-0 1-1	11 13 11
As we	We	0.00115075 1.68879e-05 0.0769231 0.0153198	1-0	869 13 1
As we	as we	0.0212766 0.00117032 0.0769231 0.00482726	0-0 1-1	47 13 1

Table 3.1: Phrase table from Moses

A distance-based reordering model is then built, and a reordering table is created. Finally, Moses decoder would make use of the language model, the phrase table as well as the reordering table. Given a source sentence, the Moses decoder will output the best translation candidate based on the scores from the above models.

3.4 R²NN SMT

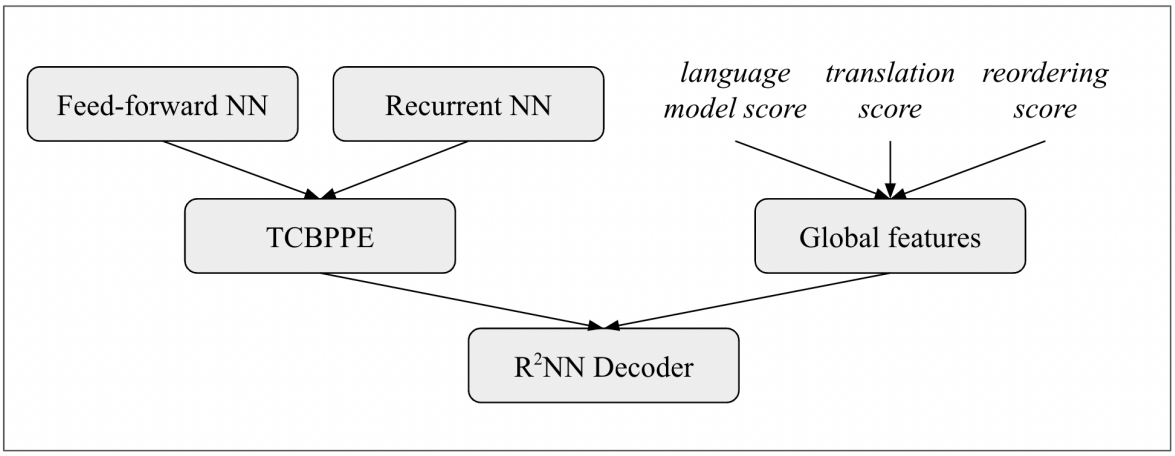

Figure 3.3: Overview of R²NN SMT

The R^{2}NN SMT consists of three parts: translation confidence based phrase pair embedding (TCBPPE), global features, and a R^{2}NN decoder.

TCBPPE

A phrase pair embedding is a vector representation which encodes the meaning of a phrase pair. Recall that our R^{2}NN model has a tree structure, where each node has a representation vector s and a recurrent vector x. The TCBPPE is to be used as the representation vector s and it is used to generate the leaf nodes of the derivation tree.

The phrase pair embedding is split into two parts: translation confidence with sparse features and translation confidence with recurrent neural network. These two vectors will be obtained separately, and will be concatenated to be used as the representation vector s in the R^{2}NN.

Global features

The global features encode global information that cannot be generated by child representations. It includes language model scores, translation scores and reordering scores which are obtained from Moses. These scores are concatenated together to be used as recurrent vectors x to the R^{2}NN model.

R^{2}NN decoder

The recursive recurrent neural network (R^{2}NN) will be used as a decoder to find the best translation candidate. For an input sentence, the decoder would construct a tree based on phrases in the sentence. An output score will be computed based on TCBPPE, global features and the structure of the tree. The translation candidate that gives the highest score will be taken as the best translation candidate.

3.4.1 TCBPPE: Sparse Features

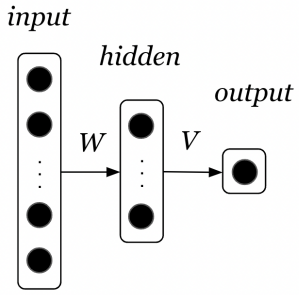

Figure 3.4: Structure of one-hidden-layer feedforward neural network

A one-hidden-layer feedforward neural network is used to get a translation confidence score using sparse features. The structure of the network is shown in Figure 3.4. Following the R^{2}NN paper[6], the size of the hidden layer is set to 20. The size of the input layer is

200,001. By training the network, we obtain hidden matrix W of size 200,001 by 20. W will be used as phrase pair embedding matrix for sparse features (`ppe_sparse`). The goal is to find the W that gives the best phrase pair embedding.

Firstly, the training data (sentences) need to be encoded as one-hot like data to be used as input to the network. The top 200,000 most frequent phrase pairs in the Moses translation table are selected, each to be a feature itself, and an additional feature is used for all the other phrase pairs. For each sentence pair (source-target) in the training data, it needs to be encoded as a vector of length 200,001 consisting of 0s and 1s, where the position i in the vector contains a 1 if and only if the i^{th} most frequent phrase pair is found in this sentence pair. An example taken from the training file:

[Source sentence] `What she would do ?`

[Target sentence] `What would she do ?`

[Encoded vector] $[\underset{0}{0}, ..., \underset{39}{1}, ..., \underset{200000}{1}]$

A 0 at position 0 means that the most frequent phrase pair does not exist in the sentence pair. A 1 at position 39 means that the 40^{th} most frequent phrase pair exists in the sentence pair. A 1 at position 200,000 means that the sentence pair contains a phrase pair that is not in the top 200,000 most frequent pairs.

index	source	target	...
0	.	.	...
...	...	...	...
39	do	do	...

Table 3.2: Top 200,000 phrase pairs

One problem with this is that the resulting vector from sentence pairs are usually sparse vectors. To save memory, instead of storing the whole vector, the position indices of 1s in the vector are stored in a file named `phrase_pair_id` (i.e. the phrase pair ids used in each sentence pair are stored). When loading the sentence pairs as training data, it is easy to obtain the one-hot encoding of each sentence pair from `phrase_pair_id`.

The next step is to obtain the expected output for each input sentence pair. Since the TCBPPE encodes translation model information, it is reasonable to use the translation scores from Moses translation model. For each sentence pair S-T containing phrase pairs $p_1, p_2, ..., p_n$, the expected output score is computed by

$$score_{S\text{-}T} = \frac{\sum_n c_{p_n}}{n}$$

where c_{p_n} is the average translation score for phrase pair p_n.

After obtaining the training dataset, we can train the neural network. The neural network will be trained for 10 epochs, where one epoch refers to one cycle of processing all the data in training set. After each epoch, the hidden matrix W is stored. The k^{th} row in W will be used as the phrase pair embedding (sparse) for the top k^{th} frequent phrase pair.

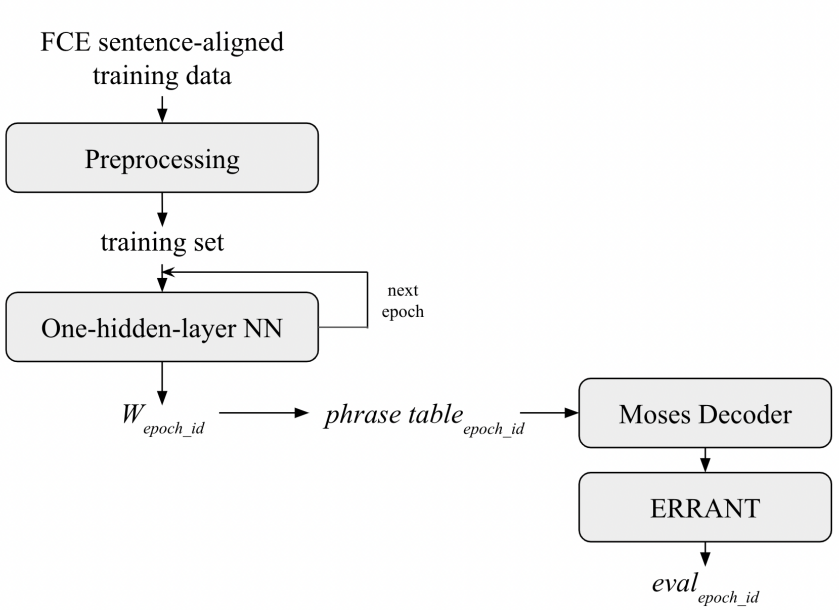

Figure 3.5: Training pipeline of one-hidden-layer feedforward neural network

That is, the embedding for the most frequent phrase pair is $W[0]$, and the embedding for phrase pairs that are not in the top 200,000 is represented by $W[200000]$.

We evaluate W by updating the Moses phrase table: for each row (ppe) in W, we add the ppe as an additional feature to the phrase table. Then we pass the updated phrase table to Moses decoder and use ERRANT to evaluate the performance. Table 3.3 and Table 3.4 shows the evaluation result by ERRANT for training epoch 5 and training epoch 10 respectively. Among all the W from epoch 1 to 10, 10-epoch W gives the highest precision, recall and $F_{0.5}$ score. Hence, W will be used as the ppe matrix for sparse features (`ppe_sparse`).

TP	FP	FN	Prec	Rec	F0.5
543	2366	4006	0.1867	0.1194	0.1677

Table 3.3: ERRANT output (5 epochs)

TP	FP	FN	Prec	Rec	F0.5
545	2363	4004	0.1874	0.1198	0.1684

Table 3.4: ERRANT output (10 epochs)

3.4.2 TCBPPE: Recurrent Neural Network (RNN)

- Word embedding (fasttext vs word2vec)

3.4.3 Global Features

3.4.4 Building R^{2}NN

- Recursive neural network (RvNN)

- Recurrent input vectors

3.5 Testing

Chapter 4

Evaluation

Chapter 5

Conclusion

Bibliography

[1] Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. The BEA-2019 shared task on grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 52–75, Florence, Italy, August 2019. Association for Computational Linguistics.

[2] Christopher Bryant, Mariano Felice, and Ted Briscoe. Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 793–805, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[3] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling, 2013.

[4] Mariano Felice, Christopher Bryant, and Ted Briscoe. Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 825–835, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

[5] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, page 177–180, USA, 2007. Association for Computational Linguistics.

[6] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. A recursive recurrent neural network for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1491–1500, 2014.

[7] Zheng Yuan. *Grammatical error correction in non-native English.* PhD thesis, University of Cambridge, Cambridge, United Kingdom, 3 2017.

Appendix A

Project Proposal