Detecting article errors in English learner essays with recurrent neural networks

Master's Thesis

Presented to

The Faculty of the Graduate School of Arts and Sciences

Brandeis University

Department of Computer Science

Nianwen Xue, Advisor

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

in

Computational Linguistics

by

Manaswini Garimella

August, 2016

Acknowledgments

I've always considered putting my ideas to words in writing on par with tooth extraction, so this thesis only happened with the help of many people.

First to my advisor, Nianwen 'Bert' Xue - thank you for the guidance, ideas and revisions on this thesis as well as all the basic concepts I learned in your classes. Thanks also to my professors James Pustejovsky, Marie Meteer, Lotus Goldberg, Keith Plaster and Sophia Malamud for laying the theoretical groundwork I came to Brandeis for. Thank you to my TAs, especially Te Rutherford, Nikhil Krishnaswamy, and Chuan Wang for adding crucial understanding outside of class.

I'd also like to acknowledge the role my Wellesley professors Stanley Chang, Andrea Levitt, and Mary Kate McGowan played for helping me develop this love of mathematics and language, and who enabled me to come to this program.

To my classmates at Brandeis - I'm glad I had such a great group of people to learn, commiserate and celebrate with, and I look forward to seeing the path each of you take.

To my friends Rebecca McGowan, Adrienne Jacobson, and Hua Chen - thank you for listening to me and putting up with my procrastination getting in the way of our plans. Now we can have fun again.

To my family - thank you Amma and Nanna for the (sometimes tough) love and life advice, and obviously all of the delicious food. Thank you Kaustubh for listening to my

complaining about computer science that you could probably do asleep. And thank you Ammamma for the unconditional love and support, and for teaching me to always push farther than I thought I could.

Finally, to David. I never would have finished this without you. You have made this past year one of the best of my life, and I can't wait for everything we'll experience together.

Abstract

Detecting article errors in English learner essays with recurrent neural networks

A thesis presented to the Department of Computer Science
Graduate School of Arts and Sciences
Brandeis University
Waltham, Massachusetts

by Manaswini Garimella

Article and determiner errors are common in the writing of English language learners, but automated systems of detecting and correcting them can be challenging to build as the context in which an article is found can be ambiguous. Here, I investigate the use of recurrent neural networks to detect and correct such errors separately, treating it as a sequence labeling task. The NUS Corpus of Learner English is used for training and evaluation. A maximum precision of 76%, recall of 41%, and F0.5 score of 46.5% were achieved in the error detection task using a bidirectional LSTM. A maximum precision of 51%, recall of 34%, and F0.5 score of 47% were achieved in the error correction task using a bidirectional LSTM, which is competitive with previous results on this corpus and recent results using other neural network models. Furthermore, the technique used relies only on lexical items, with no additional features necessary. These results, in conjunction with clear venues for improvement, show that the method is a promising one for the task.

Contents

Abstract **iv**

1 Introduction **1**
1.1 Article errors . . . 2
1.2 Overview . . . 3

2 Previous Work **4**
2.1 Machine learning approaches . . . 4
2.2 Evaluation . . . 7
2.3 The 2013 and 2014 CONLL Shared Tasks . . . 8
2.4 Recurrent Neural Networks . . . 10

3 Data **14**
3.1 Overall statistics . . . 14
3.2 ArtOrDet errors . . . 15
3.3 Training and test sets . . . 17

4 Methods **19**
4.1 Corpus Preparation . . . 20
4.2 Toolkits used . . . 22
4.3 Neural Network Architectures . . . 22
4.4 Setting Hyperparameters . . . 24
4.5 Evaluation . . . 30

5 Results **31**
5.1 Experiments . . . 32
5.2 Hyperparameter Search . . . 43

6 Conclusion **54**

Bibliography **56**

Chapter 1

Introduction

As more people learn and communicate in English, the need for tools to help English writers produce grammatically correct text grows. While grammatically correct text is not always necessary for understanding, readers often perceive grammatical correctness as a measure of competence. Text correction has wide-ranging applications, from computer-assisted language learning to post-processing steps in NLP applications such as machine translation and word prediction.

Article errors are one of the most common types of errors found in text written by English language learners, and much work has gone into developing systems to detect and correct these errors. However, these systems often use many different corpora for training and evaluation, so results are not always directly comparable. Furthermore, the systems built on well-formed text often have pipelines with tools meant for native data, which can be less effective on learner data. Language learners can also make very different types of errors based on their language background, level of proficiency in English, age, and other factors, which can be hard to generalize [21]. Thus, a system that uses original text written by learners is likely to be more effective.

Recently, recurrent neural networks (RNNs) have become very popular for various NLP tasks, including language modeling and machine translation. One of the arguments for using neural networks is that task-specific feature engineering is less necessary, as the models can automatically learn weights using raw input; however, they often require more data than other machine learning models for the same task. Since there has been very little work done in using RNNs for automated grammatical error checking, either directly or through other methods such as language modeling, this thesis makes a preliminary investigation into the use of RNNs for detection and correction of article usage errors.

1.1 Article errors

One of the most difficult areas to master for English language learners is the use of articles and determiners in English, particularly if their language background does not include any other languages that use articles. Some of the rules governing which article is used in a particular context are clear: 'a' and 'an' are used with singular, count nouns. Other rules are more opaque: what makes an instance of a noun in a particular context definite? Furthermore, some expressions are simply idiomatic [20]. For example, mountain ranges typically have 'the', e.g. 'the Himalayas' while individual mountains don't take an article, e.g. 'Mt. Everest'. Finally, the same noun may have different properties in different contexts, requiring different determiners, so a purely rule-based system is insufficient.

While language learners who are native speakers of languages with systems that are similar to that of English, such as German, might be able to intuitively decide which article is used in a noun phrase, learners from languages such as Chinese or Hindi struggle with the same decisions. Writers make three different types of article errors in a text:

- Deletion errors, where an article is missing where there should be one (typically the

most common error)

- Substitution errors, where the incorrect article is used
- Insertion errors, where an extraneous article is present

An error correction system must be able to correct all three types of errors, focusing on correcting on correcting those it can predict with high confidence rather than trying to maximize the number of errors predicted, since language learners may be less able to distinguish between good and bad predictions.

1.2 Overview

This thesis will investigate the use of recurrent neural networks for detecting and correcting article usage errors in non-native text. I use the NUCLE corpus for both training and testing, and compare results with the article and determiner prediction and correction results from teams in the CONLL Shared Tasks. Chapter 2 describes relevant characteristics of the data. Chapter 3 describes the methods used in this thesis, along with explanations of the techniques, particularly those relevant to training RNNs. Chapter 4 describes the results achieved so far along with some preliminary discussion of those results. Chapter 5 concludes with some possible avenues for future work.

Chapter 2

Previous Work

Since article and determiner errors are among the most common errors that learners make, and since they cover a small, closed class, there has been significant research in automated methods for detection and correction of the errors. Very recently, there has been effort to use neural networks for the task of grammatical error correction as well.

2.1 Machine learning approaches

Rule-based approaches were originally popular, but did not generalize well to new data. More recently, various machine learning approaches have been used to detect errors. Both supervised and unsupervised approaches using corpora of both well-formed text and learner data have been attempted, with mixed results.

Most previous machine learning approaches have focused on supervised detection and correction of errors. Few error-tagged corpora are easily available; however, some teams have developed their own annotation schemes and corpora for use with machine learning methods. Some systems have used well-formed text as training data, tagging deviations

from expectations as errors, while others use learner corpora or artificially generated errors. A common theme for all of these approaches is the feature sets they choose, which try to discover the properties of the noun relevant to classification. Most approaches also attempt to first extract noun phrases and then classify the noun phrases by the type of article necessary in each, rather than treating the entire text as a sequence.

Han, et al. (2004) use a Maximum Entropy classifier on selections from the MetaMetrics corpus, a large and diverse corpus containing both fiction and non-fiction texts from the tenth through twelfth grade reading levels [20]. While not mentioned, it is implied that the writing is error-free, or at least written by fluent speakers of English. The texts were POS-tagged, and base NPs were extracted using a heuristic-based tagger. Several features were attributed to each NP, from local word and POS context, to the head noun's countability, to the words within the NP, excluding articles if there were any. These feature sets were presented as training examples to the classifier. Test examples from TOEFL essays were constructed using the same pipeline, with the system achieving 88% and 89% agreement with human annotators. While the pipeline was fully automated, and achieved good results, the system required NPs as input rather than raw text. Because NPs have to be automatically predicted, a mistake in NP identification could result in a mistake in error correction.

DeFelice and Pulman (2008) also use a Maximum Entropy classifier trained on grammatically correct text from the British National Corpus and test the classifier on instances from the Cambridge Learner Corpus, achieving accuracies of 92.2% on correct instances and less than 10% on incorrect instances, arguing that the higher number of correct instances biases the classifier towards labeling instances as correct [12].

Turner and Charniak use an immediate-head parsing language model to classify instances based on the highest probability parse assigned by the language model. The model is trained and tested on the Wall Street Journal Corpus and up to 20 million words of the North

American News Text Corpus. The model is tested using examples from which determiners have been stripped, and achieves 86.74% accuracy. However, since it is not tested on learner data, results may not be directly comparable.

Gamon et al. (2009) use a three-step process consisting of a hierarchical MaxEnt classifier that first predicts the presence or absence of an article and then which article is required, a language model that assesses the probability of the original text and the suggested correction, and then a web search for both the original text and suggested correction. This is to ensure that precision of the suggested corrections is very high, since language learners may have difficulty distinguishing between appropriate and inappropriate corrections, and to maintain trust in the suggestion system. Their system correctly fixes errors in three different learner corpora between 64-73% of the time [14].

Since error-tagged corpora are so challenging to create, most approaches have used error-free data for training. Park and Levy (2011) use the noisy channel model and the EM algorithm to learn the noise parameters [27]. They use a combination of a bigram base language model and separate noise models for various types of grammatical errors. Each of the noise models is a weighted finite state transducer (wFST) that accepts correct sentences as input and generates sentences with a particular error type. To learn the wFST probabilities, the EM algorithm was used to train on a corpus of essays written by Korean ESL students, assuming that the sentences were the output of the wFSTs. The language model and error FSTs are composed together; to find corrections, a wFST of the observed sentence is composed with the previous model and the shortest path of the new wFST is taken as the corrected sentence. Since the method is unsupervised, Amazon Mechanical Turkers were asked to write correction versions of each sentence. The BLEU and METEOR scores, popular in machine translation, were used to assess the output from the models, comparing the corrected sentences to the original sentences. The model for spelling combined with article

errors achieved METEOR scores of .825 and BLEU score of .723, with improvement in 55 and 59 sentences and deterioration in 6 and 9 sentences respectively [27]. It is not clear how well the BLEU scores translate to other evaluation metrics more commonly used in error correction such as precision and recall, however.

2.2 Evaluation

But why is evaluation on learner data necessary? If we assume that every noun phrase, given its context, always has a single correct article, including the null article, then learner data should be no different. Models that predict which article is the correct one should be able to predict the correct article regardless of whether there is an error in the NP. However, text written by language learners, even when correct, may have different features than text written by native speakers. One problem with machine learning approaches is that features used in classification may be derived from models trained on fluent text. POS taggers and parsers trained on well-formed text without grammatical errors typically suffer in performance when run on learner data [25]. While some error detection systems have used unusual POS sequences or parse trees to detect errors, using the outputs directly as features without discriminating between incorrect and correct sequences can hide errors [25]. Furthermore, interactions between different types of errors, particularly spelling errors with syntactic errors, may make detection more difficult. There are two solutions to this problem: using learner corpora with manually annotated errors, or since manual annotation is difficult and expensive, artificially inducing errors in well-formed text.

Artificially introducing errors in text is also problematic, since the distribution of errors in learner data is not actually random [29]. Rozovskaya and Roth (2010) investigate methods of artificially introducing errors into text that will resemble the true distribution of errors in

learner data. They find that classifiers trained on data with either randomly generated errors or generated errors in the same distribution as in learner texts both outperform classifiers trained on well-formed text. For texts written by Czech speakers, a classifier trained on randomly generated text produces the best results, while for texts written by Chinese and Russian learners, they find that a classifier trained on texts with the same error distribution as the learner data produces the best results.

2.3 The 2013 and 2014 CONLL Shared Tasks

Because of the unique characteristics of learner data, some groups prefer to use learner data for both training and evaluation. A few different error-tagged corpora are freely available for research - one of the largest is the National University of Singapore's (NUS) Corpus of Learner English (NUCLE). The properties of this corpus will be discussed more extensively in Chapter 2, but approaches to article error detection using this corpus, starting with CONLL Shared Tasks using the corpus, are discussed below.

The 2013 CONLL Shared Task consisted of finding and correcting five common types of grammatical errors using the NUCLE corpus, including article and determiner errors. Precision, recall, and F1 score of each of the participating teams on the same test set was reported by the Shared Task team. The highest recall, as measured by the percentage of errors in a particular category that was corrected by the system, was achieved by the team at University of Illinois Urbana-Champagne. They follow Rozovskaya and Roth's (2010) approach, first extracting all NPs, then training a multi-class classifier with the extracted NP instances. Experimenting with different classifiers finds that an Averaged Perceptron classifier trained on these instances along with some artificially induced errors in the training data yields the best performance, with precision and recall rates of 31.99 and 59.84 respectively [28].

Other approaches by the teams participating in the shared task include language modeling, statistical machine translation, rule-based, and other machine learning approaches, with F1 scores in the range 0 to 41.69. Language modeling is a popular approach, as it has been demonstrated effective by Turner et al. in this task. Machine translation approaches are popular because both tasks can be formulated as a sequence-to-sequence learning problem. The post-processing step of translated output requires similar correction of ungrammatical text, and deals with the problem of insertion, deletion and substitution errors.

The 2014 CONLL Shared Task expanded on the original five categories to all 27 error types in the NUCLE corpus, and introduced a new test set. The teams participating in the shared tasks used many of the same methods, achieving recall percentages from 0.33%-58.85% for error correction in the ArtOrDet category. Not all of the teams reported precision, as the final output from each team only showed the suggested correction from each system. Errors may sometimes be corrected in multiple felicitous ways. Calculating recall in the task only required that the system output differed from the original text in the same locations as the gold standard. Calculating precision required that each of the system predictions that differed from the gold standard be manually evaluated for correctness since multiple corrections are possible. Regardless, we can expect that since the highest scoring team for recall, a combined team from the University of Illinois Urbana-Champaign and Columbia (CUUI), had similar results as in the 2013 Shared Task, the precision for both years by the team was similar as well.

Since I use the NUCLE corpus, with the same training and testing sets as in the CONLL 2014 Shared Task, I will use the values reported by the teams that participated as a baseline for comparison. The range of recall scores from the article or determiner errors on the official test set was 0.33 to 58.85. The team with the highest scoring recall was again CUUI, using the same approach as in 2013, with recall scores of 58.85 for article and determiner errors

in the official Shared Task test set. In their own testing, however, they report precision and recall scores of 38.73 and 10.93, which are significantly lower. The team from Cambridge, using a statistical machine translation model, reported both precision and recall of 47.31 and 49.07 on their development set, while their system's recall on the official test set was 49.48, which is much closer to the expected result.

Six out of the thirteen participating teams used some type of language model for either detection or correction of article and determiner errors, with recall ranges from 0.33 to 54.65. The team from the Pohang University of Science and Technology (POST) achieved recall of 54.65 on the official test set with a combination of finding errors through unlikely n-grams and replacing them with more likely alternatives from a language model. They do not report results for article and determiner errors separately, but do report precision and recall scores of 41.28 and 25.57 on their development set. This range of scores from all of the teams, but particularly those teams that use language modeling, is used for both the baseline and target for this thesis.

2.4 Recurrent Neural Networks

Recurrent neural networks are popular models for this type of sequence classification task, especially since they have recently proved effective at similar tasks used for automated grammatical correction, including language modeling and machine translation, with little investigation so far for the task of automated grammar correction. As in word prediction using a language model [13], article error detection can be treated as a sequence classification task in which the inputs are word vectors from the original text of the learner essays, and the outputs are classification decisions on whether an error exists at each word. As in machine translation, error correction can be treated as a sequence to sequence learning task that ac-

commodates for varying sequence lengths and a difference in the lengths of input and target sequences.

RNNs, especially their most recent incarnation in the form of Long Short Term Memory networks (LSTMs), are particularly appealing for this task for their ability to use contextual information that may be present some distance away from the error [3]. Since many factors in the context surrounding a noun phrase determine whether it requires an article, including whether the head noun is has appeared before in the text, a limited contextual window is not always sufficient for detection of an error. The following sections discuss some of the results in language modeling and machine translation using RNNs.

2.4.1 RNNs in Language Modeling

Language modeling aims to create a probability distribution of words or other language units, and is used in many areas of NLP. Creating a language model of words also allows the estimation of the probabilities of target words such as articles in a particular context, and is a popular approach for article error detection. The basic set up of most language models using RNNs use inputs of sequences of words in a sentence, often representing the words as embedded word vectors, with the outputs being a shifted sequence of words, i.e. for each input vector $< x_1, x_2, x_3, ... >$, we have the corresponding output vector $< x_2, x_3, x_4, ... >$.

Mikolov et al. (2011) train and test RNN language models on portions of the Wall Street Journal corpus, achieving perplexities from 123 using an RNN alone with the full vocabulary to 106 using the RNN with Kneser-Ney smoothing [24]. This is compared with a five-gram Kneser-Ney language model on the same training and test sets with a perplexity of 149 - significantly higher. They also investigate various architectures and methods of speeding up training time and achieving convergence more effectively, some of which are described in

Chapter 3. Further investigation into RNNs has shown they are competitive with traditional n-gram-based methods and can even outperform them [13].

2.4.2 RNNs in Machine Translation

Since statistical machine translation often includes a language modeling component, I will focus on discussing the use of RNNs for the translation alignment component instead. Recently, encoder-decoder models in which the model encodes the source sentence and decodes the target sentence which are jointly trained [33, 8]. Bahdanau et al. (2014) create a unified encoder-decoder model that aligns and translates text simultaneously using a bidirectional RNN [1]. On the WMT '14 French-English parallel corpora, they achieved BLEU scores of up to 28.85 for all words and 36.15 when unknown words were disallowed, as compared with BLEU scores of 33.30 and 35.63 respectively for a traditional machine translation model using MOSES.

2.4.3 NNs in grammatical error detection

An extension of the methods above have very recently been applied to the task of grammatical error detection. Sun, et al. (2015) set up the classification task in a similar way to previous approaches. They extract word windows around the articles $\{a, an, the, \epsilon\}$ present in the text, or around null articles at the beginnings of NPs as classification instances, and attempt to find the correct article for each window. They achieve precision and recall rates of 30.15 and 51.74 on the incorrect instances using a single-layer convolutional neural network with pre-trained word embeddings on the NUCLE corpus [31].

Chollampatt et al. (2016) have used a combined architecture a neural network global lexicon model, a feed-forward network that predicts the presence of words in a sentence,

along with a neural network joint model, a feed-forward neural network language model that predicts the word probabilities given both the source and target context [9]. These are trained on the NUCLE corpus and the Lang 8 corpus of Learner English v1.0 that has an additional 13 million tokens. Using this system, they achieve precision of 52.34, recall of 23.07 and F0.5 of 41.75 on test set of the CONLL 2014 shared task, which is higher than the combined scores of any other team. These results include all error types; breakdown by error category is not provided in the paper and includes mechanical errors such as punctuation and spelling, which are the majority of the errors tagged in the corpus.

The approach taken by Yuan and Briscoe (2016) [35] is the most similar to the one explored in this thesis. They use the encoder-decoder mechanism explained in the section above and create a bidirectional RNN model as the decoder and an attention-based decoder. They train on a subset of the Cambridge Learner Corpus, with about 28 million tokens, automatically correcting the original text. They achieve an F0.5 score of 39.90 on the CONLL 2014 test set, which is lower than Chollampatt et al., but still higher than all of the other teams that participated in the CONLL 2014 task. As with Chollampatt et al., breakdown by category is not provided in the paper.

While recent interest in neural networks has inevitably spilled over into the established task of grammatical error correction, there are still few published works in this field using these techniques. The quantity of research both in the neural network community and in the grammatical error correction community make the intersection of these tasks exciting and relevant to work on.

Chapter 3

Data

The National University of Singapore released the NUS Corpus of Learner English (NUCLE), which consists of 1414 learner essays of about 1.2 million words manually annotated for 27 different types of grammatical errors, including article or determiner errors [11]. Details of the annotation scheme and methods can be found in [11], but here I describe details relevant to this thesis.

3.1 Overall statistics

The annotation scheme allowed annotators to flag errors in 27 different categories, which were grouped into 13 categories. Annotators were required to mark the smallest ungrammatical text span, and suggest a correction that would fix the error if the original text span were replaced. While most of the essays were annotated by a single annotator, they ran a study to determine inter-annotator agreement on a 100 essay subset, finding that error correction is a challenging task.

The essays collected were written by undergraduate students who had been flagged as

requiring assistance with writing, but no additional data about their overall proficiency, native language, or other characteristics of the writers was collected. The essays were, on average, 42.34 sentences long, with 862.98 tokens and 20.38 tokens per sentence. Each document had on average 32.95 errors, and 3.82 errors per 100 words [11]. Only 1% of the errors had more than one annotation. While all of the essays were written by undergraduate students, implying a high proficiency, the error distribution in the documents was heavily skewed. Most of the essays have fewer than 25 errors while the most frequent number of errors in the essays was 15. The same is true for the number of errors in a sentence, which followed an exponential distribution - most sentences had no errors, half of the remainder had one error, half again of which had two errors and so on.

Errors are also not evenly distributed across categories, with word choice errors being the most common and local redundancies being the next most common. Indeed, one of the reasons for choosing to investigate article and determiner errors in this thesis is not only because there is extensive work in the field, but also because many of the other categories are too sparsely represented in the corpus to allow for machine learning-based approaches.

3.2 ArtOrDet errors

There are 6,004 'ArtOrDet' errors in the corpus, which places the ratio of errors per token at .5%. This category included article errors and errors in other determiners such as 'their' or 'two'. Looking only at article usage errors - those involving 'a', 'an' or 'the' in either the original text or the correction, further lowers the incidence to less than .3%. Since these essays are written by relatively sophisticated writers at the university level, they may produce fewer article errors than the average English language learner, accounting for the sparsity of article errors in the data.

The locations of each error in the text are annotated by specifying the word at which an error occurs for insertion and substitution errors. Deletion errors are indicated by specifying the word before which an error occurs. Suggested corrections are also provided for each error, including the removal of an error. Since the error spans in the text can vary between highlighting just the error itself, the entire span of text that needs to be replaced, and the error and some context, some preprocessing of the data is necessary to ensure that each error will be represented similarly. Also, the corrections provided are sometimes only suggestions when multiple corrections are possible, as indicated by various teams in the CONLL 2013 and 2014 Shared Tasks. For the purpose of this thesis, only the original provided corrections in version 2.3.1 were used.

Table 3.1 shows the distribution of the various types of article errors in the corpus. The error type is labeled by the action required to produce the correct output. The most common, accounting for more than half of all errors, are 'the' errors. This is likely because 'the' appears more frequently in noun phrases in well-formed text than 'a/an', rather than usage of 'the' being more confusing to language learners. Deletion errors are more common than insertion errors and substitution errors for the same words. Insertion errors may be more common if many of the writers' native languages don't have articles, and so the writers are likely to forget to use articles in unfamiliar contexts. There were very few instances of mixing up 'a' and 'an', since this choice is governed by a simple rule that students can learn easily. Because there are so few instances and because these types of errors can be fixed by a simple rule, they are less indicative of the overall performance of a system.

Error Type	Number in corpus	Frequency
no_error	138486	0.9599
substitute_a_an	8	0.0001
substitute_a_the	147	0.001
delete_a	162	0.0011
insert_a	659	0.0046
substitute_an_a	22	0.0002
substitute_an_the	45	0.0003
delete_an	25	0.0002
insert_an	253	0.0018
substitute_the_a	78	0.0005
substitute_the_an	13	0.0001
delete_the	1601	0.0111
insert_the	2769	0.0192

Table 3.1: Types of article errors

3.3 Training and test sets

The 2014 CONLL Shared Task gave each of the teams a training set formed from the original NUCLE corpus. A new test set was created by asking NUS students to write 50 new essays and annotating them with the original scheme. The test set was released after the conclusion of the shared task. To facilitate comparison with the results in the Shared Task, I decided to use the same training and test sets. The corpus statistics of the training and test sets can be found in table 3.2. The test set is forty times smaller than the training set, but has a much higher token and sentence error rate for article errors. The essays were written by different students and annotated by different annotators, which may explain the different error rates. Since the test set for this particular task is not representative of the training set, the task may be more similar to earlier studies in which language models using well-formed text were used to find errors, or for other machine learning models that used well-formed text as the training data.

The distribution of errors in the training and test sets is also very different - while

	Training	**Test**
Sentences	57151	1381
Types	33767	3329
Tokens	1275869	31969
Token error rate	0.45%	2.01%
Sentence error rate	8.73%	34.32%

Table 3.2: CONLL 2014 Training and Test Sets

substitution errors are the least common error type in both, over 60% of the errors in the training set are insertion errors, but deletion errors are the most common error type in the test set.

	Error Type	**Count**	**Percentage**
Training	Insertion	3681	63.66%
	Deletion	1788	30.92%
	Substitution	313	5.41%
Test	Insertion	245	38.04%
	Deletion	354	54.97%
	Substitution	45	6.99%

Table 3.3: Error types in training and test sets

A development set of the same size as the test set was chosen from the training set and used for tuning for all of the experiments detailed in the Methods and Results chapters. However, since the overall characteristics of the test and training sets were so different, a discrepancy in the results on the development and test sets could arise.

Chapter 4

Methods

I focused on detecting ArtOrDet errors in the NUCLE corpus using LSTMs. Keras, a wrapper for the Python-based deep-learning library Theano, was used to create the models [10]. The input vectors to the RNNs were word embeddings for each sentence, generated from the corpus itself. The output vectors had two nodes at each timestep, each representing the category where an error was present or not. LSTMs with a single forward layer, backward layer and bidirectional layer where forward and backward layers were concatenated were compared. Even though there has been recent success in using character-level LSTMs in language modeling and machine translation, I decided to start with the original approach of word-level LSTMs.

The training and test sets in the CONLL 2014 Shared Tasks were used for baseline comparison. For each run of the model, a validation set of the same size as the test set was randomly chosen for tuning hyperparameters.

Several different hyperparameters were varied to attempt to learn the function, including embedding size, batch size, optimization function, dropout, training time, and initialization function. To discover the optimal combination of hyperparameters, manual search was used

```
S Not only parents and teachers but also whole society pushes students
concentrating on how to get outstanding marks in examinations .
A 7 9|||ArtOrDet|||the whole society|||REQUIRED|||-NONE-|||0
```

Figure 4.1: An example sentence with a single ArtOrDet error

for each hyperparameter. Once consistent and good results were found for a particular value, that hyperparameter was fixed, and the others continued to be varied.

Since this is a sequence classification problem, categorical cross-entropy loss is used as the objective function to minimize, with a final softmax layer to normalize the output values to true probabilities. All layers in between use the tanh activation function.

4.1 Corpus Preparation

The NUCLE corpus provided error annotation in multiple formats - both XML and M2. The M2 files provided in the NUCLE corpus already identify the error extent by word indices, and were used for this thesis. As an example in figure 4.1, the error extends over both the article and the noun. Two main representations of the annotated errors were used in error detection.

The first was the approach chosen by many groups that used language modeling - to remove all articles from the texts for the input data, and using the correct articles for output data. The model would then try to learn and predict the correct article for each noun phrase in the text. Errors would be flagged if the model output differed from the writer's. One flaw of this approach, as explained by Rozovskaya and Roth (2010) [29], is that the articles themselves can be useful flags for insertion and substitution errors, and this is ignored. For example, both insertion and substitution errors are more likely to involve 'the' than 'a' or 'an'. One major advantage of this approach is that we are essentially using well-formed text

as the training data, and so we do not need an error annotated corpus. We can simply increase the size of our training data by using any additional error-free text that we think is likely to have similar distributions of articles.

The second approach was to try to guess where errors occurred directly. The inputs would then be word vectors representing each word in the original text of the essay, and the outputs would be the presence or absence of the error for the error detection task, or the appropriate correction for the error correction task. This approach had the advantage of using the possibly incorrect articles as input for determining if there was an error in that location. In Chapter 3, we see that errors occur in systematic ways, e.g. substitutions are much less likely than insertions or deletions, so the presence of an article could be used by the model to predict that an error is less likely in that location. The disadvantage of this second approach is the difficulty of obtaining additional training data. We need corpora that have been already been error-tagged. One method of getting additional training data is to artificially generate errors by deleting, adding and substituting errors in well-formed text in a similar distribution to errors in the training data we already have. This method was used in each of the following experiments to see if increasing the training data size would affect results.

To ensure that the correct representations were being used, word embeddings for the training and validation set, and finally the test set, were all trained together, as were the representations for all the output vectors. Various options for the error types were also considered - from all errors tagged as 'ArtOrDet' errors, to errors involving only articles in either the original or corrected text. To increase the number of errors in the training set, I also considered training only on sentences that had errors, but allowing all sentences with and without errors in the test set.

4.2 Toolkits used

Theano is a popular Python library for machine learning that allows for GPU computing and has been available since 2007. More recently, with the rise in popularity of neural networks, Keras, a Python wrapper for either Theano or Tensorflow, was released to allow for easy construction of neural networks [10]. Keras with a Theano back end was used for all the experiments described below, and all experiments were run on the NVIDIA Corporation GM200 [GeForce GTX TITAN X] device on an i7 processor. Some of the options for architectures and hyperparameters chosen were done so because of the options available in Keras.

4.3 Neural Network Architectures

Recurrent neural networks have been found effective at many different natural language tasks perhaps because of their unique architecture. At each time step, the model has information not only from the input, but from the output and hidden layer of the previous time step. This recurrency allows the network to keep information from arbitrarily long sequences, incorporating all necessary previous context. Unlike other neural network models that have a predetermined context, or the feature-engineering required from models like maximum emtropy or conditional random fields, the recurrent neural network can be used when we do not know how much context might be required for the model to learn the data. Most of the necessary context for determining which article is used in a noun phrase lies within the NP itself, but the NP may be many words long; furthermore, much larger context may be required, such as knowing whether the noun has already been mentioned with a particular article. These considerations make RNNs an interesting architecture to evaluate for this

task.

Since RNNs have been effective at language modeling tasks, architectures that have been used with success in language modeling were considered here. Sun et al. (2015) have found convolutional neural networks effective at making decisions on a local basis once NPs have been extracted [31], but since the errors can interact, the surrounding context can also contain errors, and since we wanted to use the possibly problematic articles themselves when making decisions, recurrent neural networks were chosen as the main architecture instead. At each time step, the model attempts to classify whether there is an error based on all of the words in the sentence and all of the previous classification decisions. Then for the input s, we have the output $f(s)$ with the conditional probabilities of each class k, here 0 or 1, $p(C_k|s)$:

$$f(s) = argmax_k p(C_k|s)$$

Given this background, we can then define the probability of a sequence of labels y for a given sentence s with length T:

$$p(y) = \prod_{t=1}^{T} p(y_t|y_1, ..., y_t - 1, s)$$

Language models trained using RNNs typically require much more text than in this corpus, which was trained only on about 800,000 words. However, since the essays contain common themes, have writers that have similar English proficiency levels and are relatively short, it seemed worthwhile to investigate RNNs for this task even with a relatively small dataset.

Although RNNs are capable of learning dependencies over long distances, they often are not able to do so in practice. Long-short term memory (LSTM) cells have recently

gained popularity because of their ability to avoid the problem of blowing up and vanishing gradients, and because they preserve information over long distance better than the original RNN cells. Keras allows for easy construction of LSTM networks with its new functional architectures. Single-direction, both backwards and forwards LSTMs were considered, along with bidirectional LSTMs.

4.4 Setting Hyperparameters

Training neural networks is often tricky because even though explicit feature engineering is not required to discover which aspects of the data are important for learning a pattern, there are many knobs that require tuning in order to learn the appropriate function that describes and generalizes the data. Here I explore a few different hyperparameters that I varied to improve performance.

4.4.1 Initialization functions

Initialization can be extremely important for training. Initializing all values to zero at the beginning may cause the network to take a very long time to learn anything, or not learn anything at all because the values are usually zero, particularly in the error detection task. Similarly, initializing all values to 1 may have the opposite effect. Initialization of all nodes to randomly selected values pulled from a particular distribution is usually suggested. Keras offers several following initialization options that were considered for this experiment, but only the following were used: 'zero', 'normal', 'glorot normal', 'he normal', 'uniform', 'lecun uniform', and 'glorot uniform', details of which can be found in Sutskever, et al. (2013) [32].

4.4.2 Loss functions

Since this is a classification problem that tries to find errors and corrections at each output, following the recommendations in Bengio, et al. (2012), [2], cross-entropy loss is used in all the experiments. That is, we have

$$E = -\sum_{i=1}^{|L|}[l_i * \log p(l_i|S) - (1 - l_i)(\log 1 - p(l_i|S))]$$

where S is the original sentence and L is the binary encoding of the label set. In the error detection case with a binary label, binary cross-entropy was used as the loss function. For the error correction case in which the labeling was not binary, categorical cross-entropy was always used.

4.4.3 Number of layers

The number of layers was also varied. All experiments with each type of LSTM were initially carried out with a single layer, and then up to four layers were added, at the lower limit for what constitutes deep learning. Since the size of the data set was fairly small in terms of LSTMs, adding more parameters in the number of layers seemed to encourage overfitting the data.

4.4.4 Hidden layer size

The size of the hidden layer was varied by using a search through binary values starting from 2 through 2048 on a log scale. As the proper interval for investigation of the size of the hidden layer was not known beforehand, for this and several of the other hyperparameters whose distribution was not known, values were chosen on a log scale. That is, initially values were

chosen from $1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048$. The upper bound was determined by computational constraints. Since the data set size was fairly small, the training process was monitored to detect overfitting.

4.4.5 Embedding size

Word embeddings have proven extremely successful in recent years for reducing the dimensionality of word vectors and simultaneously increasing representational and generalization power. Typically, training word vectors that are useful for NLP tasks requires a much larger corpus than we have here; therefore, I initially used a simple one-hot encoding for words in the corpus in the input layer and added an embedding layer during training to generate the embeddings directly. While improvements may have come from using pre-trained word vectors, like those from the Gigaword corpus, trained using GLoVE from Stanford, I was unable to find word vectors trained on a corpus of learner data. The tradeoff between memory, training time and embedding size is clear; the larger the embedding, the more memory and training time was required, so some adjustment was made purely for what would be feasibly in the space and time allowed. Like hidden layer size, manual search using values from a log scale was used, with the following values: $64, 128, 256, 512, 1024, 2048$

4.4.6 Batch size

While the most basic form of gradient descent requires the gradient for the entire dataset to be calculated before updating parameters, this can take an extremely long time and require a lot of memory. Mini-batch and stochastic gradient descent have proved effective at finding the optimum without having to calculate the full gradient at each timestep. Using too small a batch size can cause large fluctuations, while using too large a batch size can require too

much time, therefore different batch sizes were used to determine the optimum. As with embedding layer sizes and hidden layer sizes, the batch sizes were sampled from a log scale: $1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$. Larger batch sizes were too large to fit in memory and thus left out.

4.4.7 Activation functions

Keras also has various activation functions that can be used in hidden layers - here, following the recommendations in [2], we use *tanh* which can approximate any function.

4.4.8 Learning algorithms

While stochastic gradient descent is a popular learning algorithm, several learning algorithms proposed lately have been shown to be more effective at finding the optimum, either coming closer to the minimum, avoiding local minima better, or requiring less time to do so. For this thesis, using guidelines in Bengio's (2012) [2], stochastic gradient descent with adjusted learning rates, adagrad, and adamax were used.

4.4.9 Training time

Training time is one of the most important factors in finding the optimum. Stopping too early can cause the model to underfit the data, failing to fully learn the data. It may also stop before convergence is reached and the global optimum is found. On the other hand, training for too long can cause the model to overfit the data and learn noise rather than true patterns underlying the observations. Some methods to guarantee that training went for a sufficiently long time while trying to prevent overfitting are discussed below.

Early stopping

One of the best methods to prevent overfitting is early stopping of training. Rather than predefining the number of epochs to train the data and running the training process for that period of time, we monitor particular aspects of the data to see if the model continues improving. Since the training set was split into a training and validation set, after every few epochs of training, the model would predict the output on the validation set and compared with the true observations. Metrics that are often used to determine when early stopping should occur are the loss function used to train and accuracy. We stop training the model when loss on the validation set stops decreasing, or when accuracy stops increasing. Furthermore, a significant difference between the tracked metrics on the training and validation sets indicates that the model may be overfitting the data; hence we also track the difference between the desired metrics on both the training and validation sets.

As described in 4.5, accuracy is not a useful metric for this task, particularly in error detection. Therefore, loss and F1 were used as the metrics for tracking to determine when to stop training the model.

4.4.10 Training corpus size

Finally, perhaps the most important variable in the experiments was the size of the training corpus. Penn Treebank is often used as a benchmark for language modeling, with close to a million words, similar in size to the NUCLE corpus. Graves (2013) achieves perplexities using RNNs similar to more traditional language modeling techiniques on the Penn Treebank corpus, and is also able to learn structure in a corpus as large as a 100MB sample from Wikipedia [19]. Bahdanau et al. (2014) [1] use a portion of ACL WMT '14 of 348 million words to train their neural machine translation models. Since I was only using the

training portion in the CONLL shared tasks of the NUCLE corpus, my training data was far more limited in size than recommended - only about 1.2 million words. However, to test whether the performance did improve when varying the training set size, the training set was partitioned into smaller portions which were used for the experiments.

Since errors were so sparse in the data, and since the size of the training set was small to begin with, another variation that was tested was to remove all sentences in the training data that did not have errors. The test data was, obviously, left the same.

4.4.11 Meta-learning algorithm

Here we see over 10 different dimensions for learning, each of which can take many different values, giving us an exponential number of combinations of hyperparameters. Since hyperparameters interact, the optimal combination cannot be discovered by varying only a single variable at a time. Since training can often take hours, testing each of the combinations is obviously intractable. Bergstra et al. (2012) [5] show that even a grid search, while exhaustive, cannot always find the optimal combination of parameters, and furthermore, may take an extremely long time to do so. A grid search may miss the optimal value of a particular hyperparameter if it falls between the intervals used on the scale, e.g. if the perfect hidden layer size is 60 and we search from 10 to 100 in intervals of 20. Bergstra et al. (2012) [5] instead propose that a random search can be just as effective for finding the optimal combination, and do so in a much shorter period of time.

Here, I used a combination of random and manual search to find good (although not necessarily optimal) hyperparameter combinations to yield results.

4.5 Evaluation

Since there were so few errors in the text, accuracy could not be used as an informative metric. Instead, precision and recall for 'ArtOrDet' errors were used as the evaluation metrics. In the task of automatic grammar detection and correction, precision is generally favored over recall, as language learners cannot always distinguish whether an error is correctly flagged as such. The F0.5 score was used as a combined metric in the CONLL Shared Tasks as it weights precision twice as much as recall.

Since Keras only evaluated accuracy, some additional code was written to be able to evaluate precision, recall and F-score on the training and development sets during training. These were also used to evaluate the final results on the test sets.

Additionally, confusion matrices for each of the error types in the cases of both detection and correction were used for error analysis to investigate which types of errors were being caught, and whether there was a difference in the types of errors that were caught for each of the possible architectures.

The experiments were designed to simulate recent promising results in language modeling and machine translation using RNNs for article error detection, and to be comparable with previous work in article error detection, particularly those using the NUCLE corpus and in the CONLL 2013 and 2014 shared tasks. Since these experiments were run, as mentioned in Chapter 2, some other groups have had success with neural machine translation methods applied to grammar checking on the NUCLE corpus. Most of these experiments were also easily extendable to other error checking categories, and possibilities for these are discussed in Chapter 6.

Chapter 5

Results

The results for error detection were promising, and while error correction did not have results as good as those for detection, a combination of the techniques used in detection here with other techniques used by other groups for error correction may show interesting results. Error analysis shows that the errors that were most well-represented in the corpus were detected and corrected most often, an expected result. Even so, the results here are on par with the top-performing groups in the CONLL 2014 shared task, and with more recent results published by other groups using neural networks. Although all of the other groups did not provide breakdowns by error type, those that did are presented in Table 5.1 along with overall values for those that did not. All of the groups in the table used the NUCLE corpus as some or all of their training data, and reported performance on the CONLL 2014 test set.

System	Method	Error Types	P	R	F0.5
Sun et al.	Conv. NN	Articles only	30.15	51.74	32.90
Chollampatt et al.	Feed-forward	All errors	52.34	23.07	41.75
Yuan and Briscoe	LSTMs	All errors			39.90
CONLL 2014 Results					
CAMB	RB/LM/MT	All errors	39.71	30.1	37.33
CUUI	ML	All errors	41.78	24.88	36.79
AMU	MT	All errors	41.62	21.4	35.01
My results					
Maximizing F0.5	RNN	Articles only	0.5084	0.3462	0.4648
Maximizing Precision	RNN	Articles only	0.7627	0.1282	0.3833
Maximizing Recall	RNN	Articles only	0.4446	0.4117	0.4376

Table 5.1: Overall results for GEC using the NUCLE corpus

5.1 Experiments

As discussed in Chapter 4, error detection and correction required different label sets and representations for the output layer. However, similar trends were seen in both tasks, so a separate discussion on hyperparameters is at the end of the chapter, which often contains results from both tasks for the same hyperparameter.

5.1.1 Error Detection

For a simplified task, the first experiment simply considered whether errors could be detected in the essays.

This allowed for a binary classification approach. Two different input vectors were considered - inputs were word vectors using a one-hot encoding, or word embeddings. The vocabulary size was restricted to 80,000, and any out of vocabulary words were assigned a separate OOV vector. Outputs were also represented in two ways - a single layer composed of either a single node for each input with values of 0 for 'no error' and 1 for 'error', or a two-element vector, each component of which indicated the value for 'error' and 'no error'.

For the first representation, binary cross-entropy loss was used as the loss function, and categorical cross-entropy for the second. The hyperparameters described above were varied, using the search function described in the Hyperparameters section.

The inputs were then matrices with the following dimensions: $S * M * V$, where S was the number of sentences in the training set, M was the maximum length of all the sentences, and V was the size of the vocabulary. Since the model required that all of the sentences be of the same length, sentences were all zero-padded to the maximum sentence length in the corpus, with a Masking layer added. Sentences also started and ended with separate token to allow the model to learn when to start and stop learning actual inputs.

The outputs were matrices with the following dimensions: $S*M*1$ or $S*M*2$, depending on which output representation was used.

The representation we are solving for is $p(y_i|x_i, x_2, ..., x_1)$ where $y_i = 1$ if there is an error at position i and $y_i = 0$ if there is no error at position i.

Error detection, requiring only a binary output, was a much simpler task than error correction. All of the following experiments were carried out using tanh as the activation function in all of the hidden layers and cross-entropy as the loss function.

Single-layer forward LSTM

The single-layer forward LSTM was the first architecture considered. This was composed of an input layer, a single embedding layer, an LSTM layer, possibly a dropout layer, and possibly a softmax layer for the categorical cross-entropy representation.

The single layer forward LSTM showed some success in finding errors. The backward LSTM was completely unsuccessful at finding errors, which was initially surprising because it was assumed that right context is important in finding errors. The head noun of the NP is the best determiner of which article should be used, and always follows the position where

the incorrect article is, or where there should be an article. However, it may be harder for the model to predict the location of the error because, when going backwards, all of the information up to that point was correct. While the context is an important determiner of which article is appropriate in a given location, the model was attempting not to predict the correct article but rather whether there was an error in that position. The model would subsequently find errors only if there was incorrect information in the NP following the article that indicated that there was also an incorrect article. These experiments show that the article itself can be an important determiner in the task of predicting errors.

The top fifteen most common discrepancies between the model predictions and gold standard are shown in Table 5.2; the most frequent model errors involved 'the', and 'a', and show that the model tended to commit more false negatives than false positives across these common error categories, which we optimized for. The table also shows that some of the false positives occurred in impossible locations for articles, such as preceding prepositions and punctuation; discouragingly, 68% of the false positives occurred immediately before prepositions, verbs or conjunctions, or preceding the sentence start tag, even though language learners never make errors in those locations. The model may have had trouble disambiguating between different words with the same spelling such as 'can', as I did not provide any extra information beyond the words themselves and the word embeddings were trained automatically from the same small dataset.

Instead, if the representation had removed all articles and the task was to predict the correct article, the right context, including the head noun, would be much more important, as we see in the later sections.

	Bidirectional	**Forward**	**Bidirectional**	**Forward**
Token	**FN**	**FN**	**FP**	**FP**
,	0	0	18	21
.	0	0	8	3
a	47	45	5	6
ageing	11	9	0	1
an	4	4	2	0
</s >	0	0	9	4
government	14	13	0	0
human	9	10	2	1
in	0	0	7	18
of	0	0	11	11
people	1	1	7	4
population	10	11	3	1
the	330	338	12	12
to	0	0	10	6
<unk >	5	5	4	4
<s >	0	0	0	33

Table 5.2: False positives and false negatives for forward and bidirectional models

Bidirectional LSTM

The bidirectional LSTM was composed of an input layer, a single embedding layer, a forward LSTM layer that took the embedding layer as input, a backward layer that took the embedding layer as input, a layer that merged the two forward and backward layers by concatenating them, a dropout layer, and a softmax layer.

The forwards and backwards layers were repeated up to four times before the final merge, dropout and softmax layer.

The bidirectional LSTM was much more successful at discovering errors than both the single directional LSTMs. Under the various hyperparameters investigated, results varied from 0% precision and 0% recall to 76% precision and 36% recall. There did seem to be a tradeoff between precision and recall, so the F0.5 measure was used, which weights precision

twice as highly as recall. This follows the logic that an error checking system used by non-native speakers should report errors only if they are correct, as non-native speakers are probably less able to differentiate between good and bad suggestions made by a system.

Most of the entries in Table 5.2 show a similar pattern to the forward model - most of the same tokens are present in both lists. One of the main differences is that the forward model had several false positives preceding the sentence start tag <s >, whereas the bidirectional model has more false positives preceding the sentence end tag </s >, and the start tag has no errors. Otherwise, there are generally fewer false positives in the bidirectional model than in the forward model, even though false negatives remain the same. This confirms the belief that the additional context provided by the backward direction helps to disambiguate between negative and positives instances.

5.1.2 Error Correction

For the second task, error correction was explored in multiple ways. Previous attempts have involved a language modeling approach, a machine translation approach, or a classifier-based approach for each noun phrase in the text. Here, I tried both the language modeling and classifier-based approach. All of these only involved errors containing articles in the original or corrected text; other determiner errors were excluded.

For the language modeling approach, different label sets were tried. The first attempted to do a basic classification into 'deletion', 'insertion' and 'substitution' errors. The second was a much finer-grained classification of each of the three basic types with the three possible articles: 'the', 'a/an' and 'None', as in the 3.1 table in Chapter 3. For this approach, the input vectors remained the same as in the error detection approach, and only the output vectors differed. Since this was not a binary classification task, categorical cross-entropy was

used as the loss function for all experiments.

$$P(T|S) \approx \prod_{i=0}^{|T|} P(t_i|S)$$

The representation of the output vectors here gives us $p(y_i|x_i, x_2, ..., x_1)$ where $y_i > 0$ if there is an error at position i and $y_i = 0$ if there is no error at position i.

The error correction task was more difficult than the error correction task because of the increase in the size of the label set. Two different representations were used - one more coarse-grained, attempting to learning only which article belonged at a particular location, and one more fine-grained, that attempted to learn which type of error and correction were present at each location. Both tasks were further complicated by the model simultaneously learning both where articles might be present and whether there was an error at each location.

Single-layer forward LSTM

The first experiment for error correction tasks was to use a single layer forward LSTM. The inputs were fed in the order of the sentence, from left to right, with matching output vectors. Both embeddings and one-hot representations were used for the word vectors. Each possible representation of the corrections was tested with this architecture.

Bidirectional LSTM

Since the corpus is annotated for errors by the position at which an article needs to be inserted, deleted or substituted, there is often relevant context that is left out when using a left-to-right model. Most important, perhaps, is the head noun of the NP in which the error occurs, which offers clues to whether it is countable, definite or proper. A model that also takes into account the context that follows the word is then likely to be able to

make better classification decisions. Bidirectional LSTMs have been successful at building language models and in machine translation for this reason, and seemed useful for this task as well.

The bidirectional model was defined similarly with the unidirectional model, except that the results of the embedding layer were passed to two layers - one going left to right, and the other moving right to left. The results of those two layers were merged by concatenation and then fed to a softmax function. This process was repeated up to four times for a four layer bidirectional LSTM model.

Coarse-grained Representation

Many groups generated corrections looking solely at well-formed text by extracting noun phrases, removing all articles from the text, and attempting to learn the correct articles for each NP using properties of the NPs themselves. Similarly, I also experimented with a model that used inputs with all articles removed and outputs that were the correct article for each position. The Stanford parser provided with the NUCLE corpus was used to discover NPs, while a simple string match was used to remove 'the' and 'a/an'. While the same one-hot representation of the input words and and embedding layer was used, a very simple tag set was used for the output layer - $0, 1, 2$ for $'noarticle', 'the', and 'a/an'$ respectively. In the test set, grammatical errors were deduced if the model output differed from the original writer's text. While many other groups have supplemented the corpus with additional texts written by native speakers, or with unannotated well-formed text treated similarly, the distribution of articles in these texts does not always match that of learners', so I did not choose that method here. The bidirectional LSTM model was much more effective at correcting errors than a single forward LSTM model. The label set for this task required recognizing both whether an article belonged at the location and whether the article was the appropriate one.

Insertion and deletion errors can also be viewed as substitution errors involving the null article, but were treated as a separate category here. Most of the tags using this label set were 'no_error', with certain error types appearing very infrequently in the corpus. Aside from the increased size of the label set, the major difference between error detection and correction is that right context is much more important in the correction task; the head noun in the NP determines not only whether there is an error, but also what the correction should be. The left context, including the position of the article, helps determine the type of error as well, which is why bidirectional LSTMs were preferable.

The label set for this task consisted of 'no_article', 'the', and 'a/an', and the model attempted to learn the locations of each of these tags. The overwhelming majority of the tags were 'no_article', which did not differentiate between locations in which no article was possible, such as locations other than a noun phrase and locations where the null article was the correct article, such as inside a noun phrase. A post-processing step then inferred errors wherever the model's predicted output did not match the user's.

Label	**Precision**			**Recall**		
	Max	**Min**	**Mean**	**Max**	**Min**	**Mean**
no_error	0.7184	0.0070	0.2820	0.8785	0.6582	0.7604
the	0.1392	0.0442	0.1017	0.2571	0.0800	0.1748
a/an	0.1317	0.0057	0.0857	0.1982	0.0090	0.1230

Table 5.3: Precision and recall for the coarse representation

The results for this task were much lower than using the corrections labels directly, perhaps because the model was attempting to learn all of the possible correct and incorrect locations for each of the articles, rather than simply predicting the incorrect locations. Since this task was similar to language modeling approaches taken by other groups in the CONLL shared task and since RNN language models typically require much more training data than I used here, the model would probably have benefited from more data.

The confusion matrix in 5.4 shows that the majority of mistakes made by the model involved the 'no_error' tag; the model either incorrectly tagged the location as having no error when it should have, or tagged the location as having an error when it shouldn't have. There were relatively few errors in mixing up 'the' with 'a/an'. Since most of the tags were 'no_error', this is not surprising. This behavior is also similar to that of language learners - the model's predictions was similar to that of most learners, who are more likely to use 'the' than 'a/an', and omit articles instead of using extraneous articles.

		Original		
		the	**a/an**	**no article**
Gold	**no article**	47045	317	37
	the	164	474	649
	a/an	81	30	189
		Predictions		
		the	**a/an**	**no article**
Gold	**no article**	46487	616	296
	the	393	237	19
	a/an	200	40	60
		Predictions		
		the	**a/an**	**no article**
Orig	**no article**	46420	579	291
	the	518	278	25
	a/an	142	36	59

Table 5.4: Confusion matrices between gold tags, original text, and predictions

Fine-grained correction representation

As seen in Table 5.5, the performance was better for error types that were well-represented in the corpus. Since the overwhelming majority of the tags were 'no_error', the precision and recall for those positions was the highest. Indeed, as in the error detection case, several of the model runs tagged all of the outputs as not containing an error, which yielded the perfect recall in the table. Errors involving 'the', which were the most common article error

Label	Number	Precision			Recall		
		Max	Min	Mean	Max	Min	Mean
no_error	138486	0.9912	0.9848	0.9889	1.0000	0.8177	0.9956
substitute_a_an	8	0.3333	0.0000	0.0354	0.1818	0.0000	0.0197
substitute_a_the	147	1.0000	0.0000	0.0565	0.1515	0.0000	0.0232
delete_a	162	0.5714	0.0000	0.0892	0.2424	0.0000	0.0535
insert_a	659	1.0000	0.0000	0.0179	0.5000	0.0000	0.0117
substitute_an_a	22	0.2500	0.0000	0.0009	0.1250	0.0000	0.0004
substitute_an_the	45	0.7273	0.0000	0.1976	0.4211	0.0000	0.1038
delete_an	25	0.4000	0.0000	0.0447	0.2500	0.0000	0.0310
insert_an	253	0.0233	0.0000	0.0001	0.0526	0.0000	0.0002
substitute_the_a	78	1.0000	0.0000	0.3197	0.2871	0.0000	0.1072
substitute_the_an	13	0.6667	0.0000	0.2944	0.3232	0.0000	0.1705
delete_the	1601	0.4326	0.3991	0.4151	0.2808	0.1924	0.2282
insert_the	2769	0.3007	0.2165	0.2622	0.3598	0.2622	0.3191

Table 5.5: Max, min and mean precision and recall for fine-grained corrections

types, also yielded good and consistent results, with performance comparable to the simpler error detection task.

When taken collectively, these results are comparable to some of the better results in the CONLL 2014 shared task, with a maximum precision of 0.44 and recall of 0.32, and average precision and recall of 0.25 and 0.22 respectively. Since many of those teams did not report precision, these results seem very competitive. These results would have to be further compared using other metrics such as BLEU with other groups doing neural error correction, but show that it might be possible, especially if the size of the training corpus was increased, to be able to directly predict where an error occurs and the necessary correction directly without using other methods such as language modeling or machine translation.

POS tags

Since articles should always be found in NPs, and since POS tags may offer some generalization that the words themselves do not, POS tags are a common feature used in error correction systems. The final experiment used a model that included not just the word embeddings for the words in each sentence, but vectors representing the POS tags as well. The sentences were tagged using the 2016 Stanford POS bidirectional-distsim pre-trained tagger. Even though other groups have found that POS tagging performance suffers when, first, the tagged text is in a different domain from the text the tagger was trained on, and second, when the tagger is trained on well-formed text whereas the target text is learner data, since the incidence of errors is so low in the text overall, at least 95% of the NPs in the text would have been tagged correctly using the POS tagger. Furthermore, incorrect POS tags or unexpected tags, such as the presence of an article before a word tagged as a verb, maybe reliable indicators of whether there is an error in a particular location.

The word vectors and the POS vectors were concatenated for both the forward and backward directions in a bidirectional LSTM model. The forward and backward layers were merged before an optional dropout layer was included, and ending with a softmax layer to normalize the output to true probabilities. Unfortunately, there wasn't enough time to sufficiently tune these models to achieve the best performance. However, there was evidence to show that POS tags were helpful in raising recall. Table 5.6 shows that precision was low for all categories, especially when compared with the model in Table 5.5 that did not use POS tags as an input layer. This was especially surprising because I had assumed that POS tags would provide additional information and further context, therefore increasing precision.

	Precision			Recall		
Labels	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**
no_error	0.9970	0.9863	0.9957	0.9997	0.2250	0.9075
substitute_a_an	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
substitute_a_the	0.5000	0.0000	0.0207	0.1818	0.0000	0.0284
delete_a	0.2000	0.0000	0.0359	0.1818	0.0000	0.0477
insert_a	0.2000	0.0000	0.0342	0.1935	0.0000	0.0773
substitute_an_a	0.3333	0.0000	0.0126	0.2500	0.0000	0.0156
substitute_an_the	0.0625	0.0000	0.0007	0.1250	0.0000	0.0013
delete_an	0.5000	0.0000	0.0281	0.5000	0.0000	0.0443
insert_an	0.4615	0.0000	0.1073	0.4211	0.0000	0.1436
substitute_the_a	0.2222	0.0000	0.0167	0.2727	0.0000	0.0360
substitute_the_an	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
delete_the	0.2650	0.0000	0.1819	0.3281	0.0000	0.1762
insert_the	0.3605	0.0009	0.0745	0.3537	0.1829	0.2572

Table 5.6: Precision and recall for model with POS tagged layer

5.2 Hyperparameter Search

The following hyperparameters were considered for all of the tasks above, and generally exhibited the same trends, regardless of task, unless specially noted below.

5.2.1 Training time

Training time is considered one of the most important factors in determining whether the algorithm has converged. Stopping training too early means the model has not yet learned the data, while training for too long can mean that the model has overfit the training data, learning noise present and will not be able to generalize well when tested. The results below show the precision, recall and F0.5 scores for the same model stopped at different times.

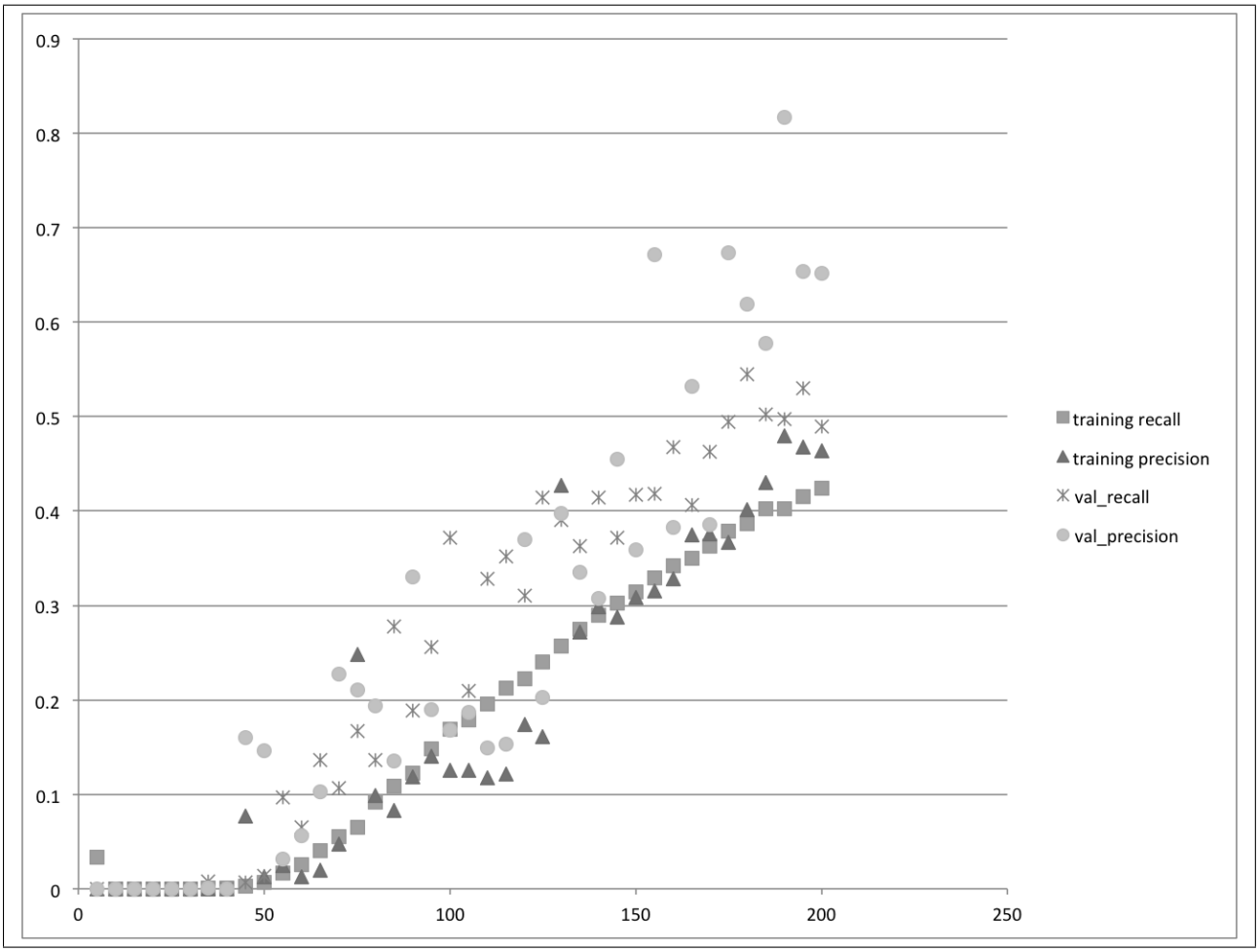

Figure 5.1: Precision and recall for training and validation sets over time

Early stopping

A validation set chosen from the training set that was the same size as the CONLL 2014 test set was used for verifying whether the model was overfitting during training. Periodically, loss and accuracy on the training and validation sets were measured. If the training loss and development loss were similar and continuing to decrease, training was continued to prevent underfitting. If the training loss was continuing to decrease while the validation loss stopped decreasing, training was stopped and the model was assumed to be overfitting the

data. However, as shown in Table 3.2, there are many more errors in the test set than there were in the training set, and the distribution of error types was different as well, so using a validation set chosen from the training set may not have been the best indicator of whether the model was overfitting.

As seen in Figure 5.1, the results were often better on the validation set than on the training set, for a given number of epochs, so it did not appear as though the model was overfitting. However, the results on the test set after training a model were always worse in terms of both precision and recall than they were on the training and validation sets. The mean difference between the F0.5 scores of the validation set and the test set at the end of training, was 0.54. This difference was consistent regardless of the hyperparameters used, indicating that there was too great a difference in the distribution of the errors in the training and test sets. Since I was attempting to compare my results with other results in the CONLL 2014 shared task, I used the same test set, but it seemed clear that the models were always learning some noise and overfitting during training.

5.2.2 Initialization functions

Initialization had a significant impact on the overall performance of the models, as shown in Table 5.7. Glorot normal initialization [16] achieved the highest precision and recall results, as well as the highest mean precision across all model runs. However, it also had some of the worst performance in both precision and recall. Model results using initializations drawn from uniform distributions were more consistent, with a smaller range in both precision and recall than those using normal distributions. As predicted from [2], zero initialization had some of the worst results, particularly in recall, as the model predicted very few errors. The sparsity of the errors in the text may have required more initial weight to be assigned during

initialization .

Since errors were so sparse in the data overall, even though the networks investigated here were not particularly deep, initialization was especially important. Early results before fine-tuning showed all outputs going to zero, since over 98% of the initial y-values were zero, i.e. 'no error', which is most pronounced when using zero initialization. It was important, then, to seed the network with inputs that would counteract this effect until the network started to learn the errors.

Initializations	**Precision**			**Recall**		
	Max	**Min**	**Mean**	**Max**	**Min**	**Mean**
glorot_normal	0.7627	0.0011	0.3076	0.4117	0.0014	0.1722
glorot_uniform	0.3610	0.0529	0.2033	0.2308	0.1368	0.1969
he_normal	0.6154	0.0005	0.2304	0.2593	0.0028	0.1348
he_uniform	0.3208	0.0990	0.2472	0.2521	0.1339	0.2052
lecun_uniform	0.6087	0.0064	0.2856	0.2778	0.0100	0.1380
uniform	0.4590	0.1060	0.2518	0.3746	0.1439	0.2036
zero	0.3023	0.0075	0.1832	0.2308	0.1410	0.1906

Table 5.7: Max, min and mean values for each initialization function

5.2.3 Number of layers

The number of layers in each model made a significant difference, with additional layers improving precision. Recall stayed the same, with a slight drop when only two layers were used. The results were also more consistent, with a smaller range between the maximum and minimum with four layers than with two or three layers. The range was the smallest with a single layer, but both precision and recall were very low with a single layer.

Number of layers	Precision			Recall		
	Max	Min	Mean	Max	Min	Mean
1	0.1590	0.1128	0.1348	0.3576	0.1510	0.2125
2	0.7627	0.0005	0.2653	0.4117	0.0028	0.1628
3	0.7500	0.2837	0.3909	0.3433	0.0043	0.2099
4	0.7564	0.3734	0.5197	0.3989	0.0798	0.2169

Table 5.8: Max, min and mean values for precision and recall with varying number of layers

5.2.4 Hidden layer size

The size of the hidden layer can also affect performance, and no prescribed formula exists for determining the appropriate number of nodes, although there are heuristics to help estimate. A similar approach was used in all of the numerical hyperparameters; to use intervals from a log scale to help determine the optimal value. Table 5.9 shows dramatically better results for both precision and recall at a layer size of 128 nodes than at smaller or larger layers.

Hidden Layer	Precision			Recall		
	Max	Min	Mean	Max	Min	Mean
32	0.1863	0.0000	0.0689	0.0966	0.0000	0.0364
64	0.1644	0.0000	0.0533	0.1006	0.0000	0.0408
128	0.7627	0.0000	0.2612	0.4117	0.0000	0.1727
256	0.3112	0.1206	0.1752	0.2350	0.1610	0.2086

Table 5.9: Precision and recall for hidden layer size

5.2.5 Embedding size

The embedding size did not seem to be a major factor in performance, perhaps because the size of the set was so small. It seemed to influence training time more than having an effect on the precision or the recall. The minimum precision and recall did improve as the embedding size increased, suggesting that the model was learning some useful contextual information in the embedding layer that allowed the models to have a lower bound in performance. The

maximum and mean values did not increase, however, suggesting that there was an upper bound to how much semantic information could be extracted from a limited corpus using this method.

Embedding Size	Precision			Recall		
	Max	Min	Mean	Max	Min	Mean
128	0.9964	0.0000	0.5221	1.0000	0.0000	0.5009
256	1.0000	0.0000	0.5051	1.0000	0.0000	0.5303
1024	0.9966	0.0000	0.5714	1.0000	0.0000	0.5368
2048	0.9969	0.0000	0.6038	1.0000	0.0000	0.5520
4096	0.9962	0.1351	0.5661	0.9944	0.1866	0.5943
8192	0.9962	0.1394	0.5694	0.9950	0.1709	0.5931

Table 5.10: Precision and recall at different embedding layer sizes

5.2.6 Batch size

Since this is a relatively small dataset, using full batches or larger batches is recommended, as using a very small batch to calculate a gradient may not be representative of the full dataset, and the loss function may show large fluctuations. In practice however, I did not have enough computational power to compute gradients using the complete dataset, so batch sizes smaller than the complete data set were used. As explained in [2], the batch size is less important to the final performance of the model and is more relevant to training time. A particular batch size coupled with the appropriate training time should not affect overall performance. In Table 5.11, shows better precision for a batch size of 128, and better recall for a batch size of 64. The performance of both metrics drops off as the batch size increases; the training time may not have been adjusted appropriately for each batch size, and thus we see overfitting in the models with higher batch sizes.

Batch size	Precision			Recall		
	Max	Min	Mean	Max	Min	Mean
64	0.4178	0.0033	0.1901	0.2365	0.1268	0.1779
128	0.7627	0.0000	0.1559	1.0000	0.0000	0.1022
256	0.6667	0.0000	0.2255	0.2835	0.0000	0.0766
512	0.6667	0.0000	0.2459	0.2165	0.0000	0.0370
1024	0.4800	0.0000	0.0478	0.0171	0.0000	0.0011

Table 5.11: Precision and recall with different batch sizes

5.2.7 Optimizing algorithms

The choice of learning algorithm greatly affected the results. Keeping the other hyperparameters constant for the bidirectional LSTM for error detection, some optimizers yielded similar results every time, while others varied greatly. Figure 5.2 shows the differences in the precision-recall curve in results between the optimizing functions tried. The results for stochastic gradient descent (sgd) varied greatly, but also produced some of the best results, with higher recall and precision. In general, sgd showed better precision than recall. This is consistent with the advice in [2], which argues that sgd requires more tuning with careful adjustments of learning rate and regularization. Setting a learning rate that is too high may cause the algorithm to diverge, so a learning rate adjustment of 0.1 was applied with Nesterov momentum, but training may not have been long enough for the algorithm to converge. Further tuning was not explored, which may explain why the results were so disparate - the model may not have been learning the data sufficiently or reach the optimum, so with random initialization, we see different results with each run.

To prevent these problems of failing to find the optimum and adjusting the learning rate, other optimizers have been defined. RMSProp is a method that keeps a moving average of the squared gradient for each weight, and thus works well for adapting the parameters separately. The results here, however, were not much different with RMSProp, perhaps

because the size of the dataset was still small, portions of the data had very different error distributions, and the batch sizes were large; the precision fluctuated greatly and recall was very low overall.

Adagrad has a per-parameter learning rate, which makes it a better algorithm for sparse parameters, as those which are represented more often are adjusted more frequently, and those that are represented less often are not adjusted as often. This allows the algorithm to converge more quickly than sgd. Since the article errors in the data set were sparse and often very different, this may explain why the recall was higher than when using RMSProp or sgd. The consistent results and higher recall may also reflect that these runs were converging, unlike those with sgd, RMSProp and Adadelta.

For Adam and Adamax, recall was similarly fairly consistent between 0.15 and 0.25, but precision varied between 0 and 0.3. These two highly related algorithms show very similar results in the graph, which is to be expected. Both are also related to Adagrad, which may explain why the recall was similar, even if precision varied more.

Overall, however, there was a large range in the precision, but recall stayed low. There seemed to be a clear tradeoff between precision and recall, as the highest precision results also had very low recall, and the highest recall results had very low precision.

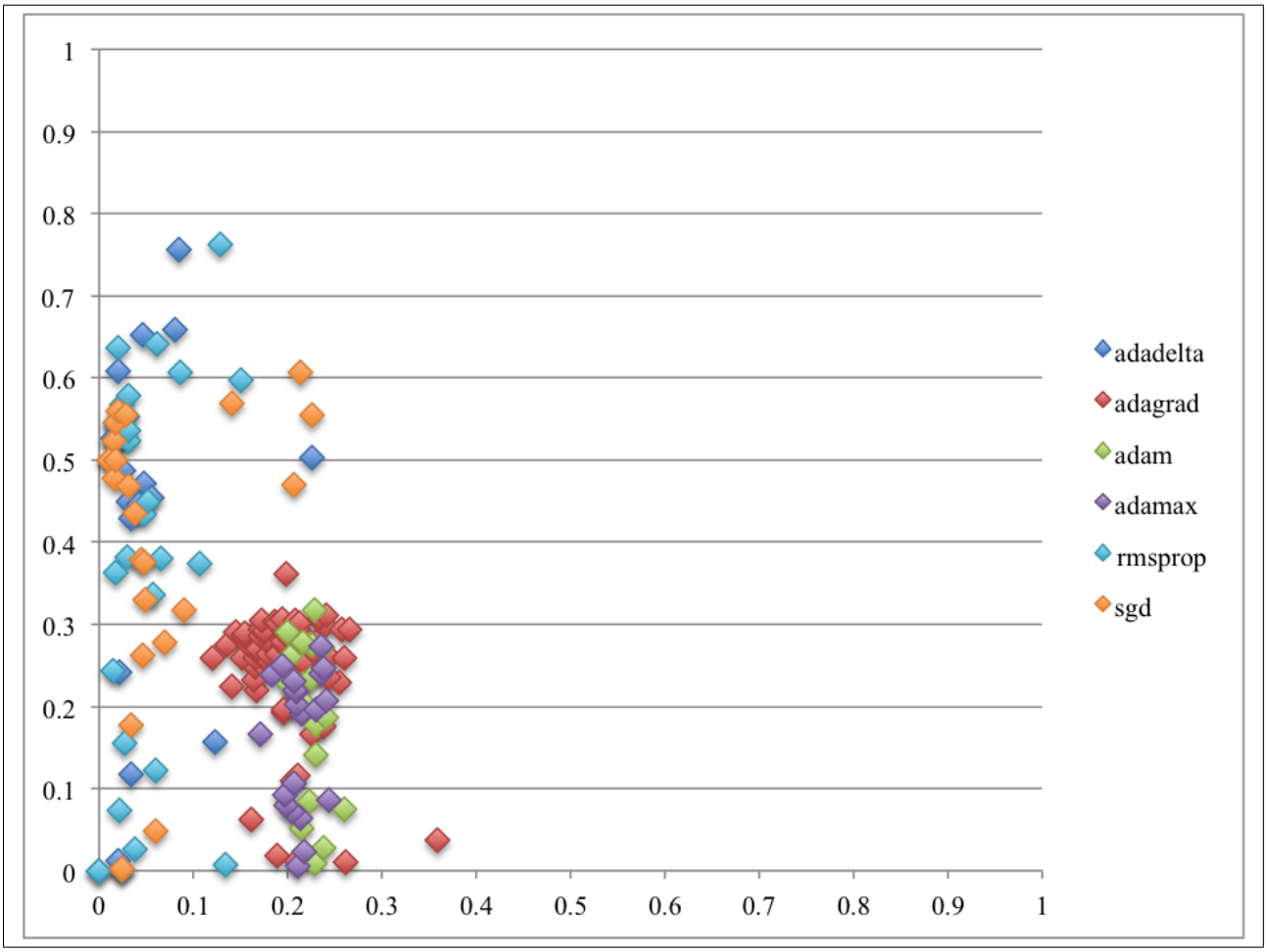

Figure 5.2: The precision-recall curve for different optimization functions

5.2.8 Training corpus size

Since I was dealing with a relatively small training corpus, it seemed likely that the size of the corpus would be a limiting factor, especially with error recall. On the other hand, the corpus may have been highly redundant, with the same types of errors appearing repeatedly, in which case reducing the size of the corpus would not have a profound effect on performance. This proved to be false when reducing the size of the training set from 50,000 sentences to 30,000 sentences. Both the precision and recall plummeted to less than a fourth when the size was

reduced; it seems likely that increasing the size of the training set for this task would have benefits for the performance of the model. Since models using LSTMs are typically trained on much larger datasets, this result is not surprising. More data would allow the model to distinguish between environments that were likely to have errors from those that would not. Furthermore, since the error distribution in the test corpus differed so significantly from that in the training corpus, reducing the size of the training corpus meant that even fewer examples of errors were present in the training data, reducing precision especially.

Number of sentences	**Precision**			**Recall**		
	Max	**Min**	**Mean**	**Max**	**Min**	**Mean**
30000	0.1863	0.0	0.0614	0.1006	0.0	0.0386
50000	0.7627	0.0	0.2096	0.4117	0.0	0.1141

Table 5.12: Precision and recall for different training set sizes

The various models and hyperparameters explored here yield a consistent pattern: precision was generally easier to achieve than recall, which was expected given the small corpus size and and since RNNs typically require massive datasets before they outperform traditional machine learning methods. They confirm results from previous RNN experiments on similar tasks such as machine learning and language modeling, with expected effects of the various hyperparameters. Since the search space was so large, a meta-learning algorithm that found the appropriate combination of hyperparameters, such as random search, would have helped. Even though constant intervals were not used for the numerical hyperparameters, some of the intervals were very large and could have used further tuning. The most revealing experiments here show the dramatic effects the size of the corpus have on performance of both precision and recall of different error types, and that the test set was sufficiently different from the training set that generalizing from the small training set was a challenge. Both of these difficulties, along with the relative sparseness of the grammatical errors in the

overall text affected the quality and robustness of the results from the experiments.

Chapter 6

Conclusion

Article error detection and correction has been studied extensively but is far from a solved task. The best models tested on learner corpora are still not sufficiently robust to be able to be used by learners in a classroom. Recent advances in the use of neural networks allows us an opportunity to test new methods for error detection and correction. Even shallow networks such as those used in this thesis are promising; the results here in this particular error category rival the best performing models in the CONLL 2013 and 2014 shared tasks that used more traditional models. More advanced networks with additional components, such as those currently used in machine translation, may prove to be even more effective at this task.

The results shown here can be extended and improved in a number of ways, most importantly by using a larger training set. Negative information in the form of correct sentences written by language learners can help improve precision. Sentences written by learners with a lower English proficiency than those who contributed to the NUCLE corpus may have more errors, which might help boost recall. The error analysis shows that additional information is required to find substitution errors especially in the training set. Introducing noise

by changing some of the correct articles randomly would increase the number of positive training samples, and is a technique used with success in image processing. Even if the changed samples do not match the initial distribution, the model would be more likely to learn that such errors occur at all. Also, some more experimentation with hyperparameters, particularly more rigourously implementing early stopping to determine training time, training corpus size, and initialization, may continue to help improving performance. Algorithms for automatically determining hyperparameters for a given task can also reduce the labor necessary in manually identifying these features.

The final step is to use the positive results here for discovering grammatical errors in other categories. Article error detection is well-studied, but verb errors, another extremely common error made by language learners, has been given less consideration in the literature. Verb errors are also well suited to be learning using RNNs because there are a wide variety of error types that occur, from verb tense errors to verb form errors, and many of the error types have long distance dependencies. Indeed, verb tense error detection often requires information from previous sentences or previous paragraphs, and a temporal narrative. Like article errors, local information, such as the incorrect form used by the writer, can be useful in detecting errors. The ability of LSTMs to learn long range dependencies while paying attention to information at hand is very promising in this respect. If the transition to different error types proves successful, the techniques may also be used for grammatical error correction in languages other than English.

Error detection has applications outside language learning as well - since the types of errors a writer makes reflect their age, proficiency, dialect, and native language, the output of these models could create a profile of a particular writer that can be used for native language identification or author identification generally, and RNNs may be an effective method for discovering those patterns.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[3] Yoshua Bengio, Nicolas Boulanger Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE, 2013.

[4] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer, 2006.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[6] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[7] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[8] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[9] Shamil Chollampatt, Kaveh Taghipour, and Hwee Tou Ng. Neural network translation models for grammatical error correction. *arXiv preprint arXiv:1606.00189*, 2016.

[10] François Chollet. Keras, 2015.

[11] Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. Building a large annotated corpus of learner english: The nus corpus of learner english. In *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 22–31, 2013.

[12] Rachele De Felice and Stephen G Pulman. A classifier-based approach to preposition and determiner error correction in l2 english. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 169–176. Association for Computational Linguistics, 2008.

[13] Wim De Mulder, Steven Bethard, and Marie-Francine Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, 2015.

[14] Michael Gamon, Claudia Leacock, Chris Brockett, William B Dolan, Jianfeng Gao, Dmitriy Belenko, and Alexandre Klementiev. Using statistical techniques and web search to correct esl errors. *Calico Journal*, 26(3):491–511, 2009.

[15] C Lee Giles, Steve Lawrence, and Ah Chung Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine learning*, 44(1-2):161–183, 2001.

[16] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[18] Alex Graves. *Supervised sequence labelling*. Springer, 2012.

[19] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[20] Na-Rae Han, Martin Chodorow, and Claudia Leacock. Detecting errors in english article usage with a maximum entropy classifier trained on a large, diverse corpus. In *LREC*. Citeseer, 2004.

[21] Ekaterina Kochmar. *Identification of a writers native language by error analysis*. PhD thesis, Masters thesis, University of Cambridge, 2011.

[22] Claudia Leacock, Martin Chodorow, Michael Gamon, and Joel Tetreault. Automated grammatical error detection for language learners. *Synthesis lectures on human language technologies*, 7(1):1–170, 2014.

[23] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[24] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[25] Ryo Nagata, Edward Whittaker, and Vera Sheinman. Creating a manually error-tagged and shallow-parsed learner corpus. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1210–1219. Association for Computational Linguistics, 2011.

[26] Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. The conll-2014 shared task on grammatical error correction. In *CoNLL Shared Task*, pages 1–14, 2014.

[27] Y Albert Park and Roger Levy. Automated whole sentence grammar correction using a noisy channel model. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 934–944. Association for Computational Linguistics, 2011.

[28] Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, and Dan Roth. The university of illinois system in the conll-2013 shared task. In *In Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared*. Citeseer, 2013.

[29] Alla Rozovskaya and Dan Roth. Training paradigms for correcting errors in grammar and usage. In *Human language technologies: The 2010 annual conference of the north american chapter of the association for computational linguistics*, pages 154–162. Association for Computational Linguistics, 2010.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[31] Chengjie Sun, Xiaoqiang Jin, Lei Lin, Yuming Zhao, and Xiaolong Wang. Convolutional neural networks for correcting english article errors. In *Natural Language Processing and Chinese Computing*, pages 102–110. Springer, 2015.

[32] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.

[33] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[34] Jenine Turner and Eugene Charniak. Language modeling for determiner selection. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 177–180. Association for Computational Linguistics, 2007.

[35] Zheng Yuan and Ted Briscoe. Grammatical error correction using neural machine translation. In *Proceedings of NAACL-HLT*, pages 380–386, 2016.