

NOISING AND DENOISING NATURAL LANGUAGE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Ziang Xie
December 2018

© 2018 by Ziang Xie. All Rights Reserved.

Re-distributed by Stanford University under license with the author.



This work is licensed under a Creative Commons Attribution-Noncommercial 3.0 United States License.

<http://creativecommons.org/licenses/by-nc/3.0/us/>

This dissertation is online at: <http://purl.stanford.edu/zc799jh3207>

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Andrew Ng, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dan Jurafsky, Co-Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Silvio Savarese

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Written communication is a crucial human activity, and one that has increasingly been altered by technology, including Artificial Intelligence. In this thesis, we describe an application of AI to build computational writing assistants that suggest edits to text, with the aim of correcting errors, improving fluency, and modifying style.

Existing AI writing assistants are highly limited in their ability to provide useful suggestions due to the challenge of data sparsity. When state-of-the-art neural sequence transduction models are fed inputs that do not match the training distribution, low-quality and noninterpretable outputs often result. Thus such models must be trained on large parallel corpora—often on the order of millions of sentence pairs—to attain good performance, and even then, they continue to improve with more data.

With the end goal of developing more effective AI writing assistants, this thesis addresses the challenge of data sparsity by investigating the effects and applications of noise in the sequence modeling setting. We begin with a theoretical analysis of the effects of sequence-level noise that illustrates the insufficiency of existing approaches for understanding and modeling such noise. With this understanding in place, we develop a method for synthesizing more diverse and realistic noise in natural language, thus remedying the need for parallel data for the task of “denoising” or suggesting edits to writing. To demonstrate the broader applicability of this method, we describe an extension to generate stylistic edits without parallel data between different styles. We close by describing an AI writing assistant that we deployed to validate the methods proposed in this thesis, along with findings to improve such AI assistants in production.

For my parents.

Acknowledgments

This thesis is the result of close collaborations with my advisers and peers. First, I am extremely fortunate to have worked with my advisers, Andrew Ng and Dan Jurafsky.

Like millions of others who were first introduced to machine learning through Andrew’s course materials, I first learned of machine learning through the Stanford Engineering Everywhere videos that Andrew made available to learners worldwide. There is not a single subfield of machine learning that has not been impacted by Andrew, and his advice and support during my PhD have been humbling and invaluable. When encountering almost any problem in machine learning, my thoughts usually fall to some conversation with Andrew—not just on technical aspects, but on how to select what piece of the problem to examine next, and even whether I’m approaching the right problem at all.

Ask anyone about their experience working together with Dan, and invariably they will brighten up and begin an outpouring of praise and gratitude. His empathy and kindness are exhibited not just through personal interactions, but in the focuses of his work, from his work on social meaning, to his research and book on the language of food—ketchup, who knew!—to his devotion to teaching and language education. Recently, I was struck by how some of the things that make me (along with, granted, probably the majority of people in the world) happiest—for example food or books—are areas that Dan has done a lot of work in. That really blew me away.

The start of my career is due to my undergraduate adviser at UC Berkeley, Pieter Abbeel. Besides his unbelievable, contagious exuberance, Pieter has always been incredibly generous with his time. Many of my fondest memories of Berkeley are those of my chatting with Pieter, where, even a year into us working together, I

always remained flabbergasted by his willingness to put so much energy into my success and the success of his other students. I would not be in machine learning today if it was not for him, and I miss working in the Robot Learning Lab with him dearly.

This thesis has also benefitted from the guidance of several other faculty and mentors. Noah Goodman’s advice on the nuances of language has shaped much of my recent thinking about challenges in NLP. Chris Potts’ teaching and work on natural language understanding is a reminder of exciting work to be done that this dissertation, unfortunately, does not address. James Zou’s broad interests in ML theory, NLP, and genomics has inspired me to look at a wider range of problems.

Silvio Savarese has given several rounds of feedback on my research, almost immediately identifying promising directions from the refreshing vantage of an expert in vision and robotics. I also thank Silvio and Percy Liang for allowing me to rotate with their groups during the first year of my PhD. It took me a while to onboard after starting my PhD, and both Percy and Silvio were incredibly patient with me those quarters. Quoc Le hosted me during a summer internship at Google Brain, and it was a privilege learning from his unique ability to combine seminal research with systems that impact hundreds of millions of people.

Many peers helped and supported me in the past five years. In the Stanford ML group, there are many peers that I have had the privilege of working together with or learning from. Adam Coates, Sameep Tandon, Brody Huval, Tao Wang, Joel Pazhayampallil, Will Song, and Jeff Kiske, thanks for the fun collaboration on self-driving. Andrew Maas, Peng Qi, and Awni Hannun, for all the help getting up to speed and working on speech. Anand Avati and Naveen Arivazhagan were there from the beginning of the work that led to this thesis, and shortly after I was fortunate to work with Daniel Levy and Allen Nie on a project that led to another chapter of this thesis. Other friends and collaborators I thank include Misha Andriluka, Pranav Rajpurkar, Sharon Zhou, Hao Sheng, Swati Dube, Ferdinand Legros, Prateek Verma, Dillon Laird, Jeremy Irvin, Daisy Ding, Aarti Bagul, Albert Haque, Michelle Guo, Tony Duan, and Cindy Wang.

Members of the Stanford NLP group have been incredibly supportive as well,

including Thang Luong, Danqi Chen, Siva Reddy, and Will Monroe. In particular, I thank Jiwei Li and Sida Wang, for going out of their way to work together on the noising paper together, as well as for the fun times hanging out. Other Stanford friends I would like to thank include Zayd Enam, Sathish Nagappan, Samir Menon, Jayesh Gupta, Jonathan Ho, and Saumitro Dasgupta. I also thank Joseph Lee, Nathan Dalal, Christina Pan, Guillaume Genthial, Max Drach, Stanley Xie, and Jason Jong, who worked together on the deployment piece with—research that many may find too engineering-heavy.

Thank you to all of the technical and administrative staff who have helped me at Stanford, including the Stanford Action team, for keeping our cluster going strong, and the PhD Student Services team, for helping me navigate through this 5+ year process.

Outside of Stanford, I thank Roberto Lopez, Nicholas Solichin, Jason Jong, Kaitlyn Menghini, and Arjun Singh for their friendship. In particular, I’ve been very lucky to have Steven Tan as an incredibly thoughtful and encouraging friend.

Finally, I thank my parents and family for their love and support. This dissertation is dedicated to them.

Contents

Abstract	v
Acknowledgments	vii
1 Introduction	1
1.1 Contributions by Chapter	3
1.2 First Published Appearances	5
2 Background	7
2.1 Setting	7
2.2 Neural Sequence Transduction Models	9
2.3 Attention	11
2.4 Training Overview	13
2.5 Decoding Overview	13
2.6 Evaluation	16
2.7 Other Topics	16
2.8 Conclusion	17
3 Denoising Natural Language	18
3.1 Model Architecture	19
3.1.1 Character-Level Reasoning	20
3.1.2 Encoder Network	21
3.1.3 Decoder Network	21
3.1.4 Pyramid Structure	22

3.2	Decoding	22
3.2.1	Language Model	23
3.2.2	Beam Search	23
3.2.3	Controlling Precision	23
3.3	Experiments	24
3.3.1	Training and Decoding Details	25
3.3.2	Noisy Data: Lang-8 Corpus	25
3.3.3	Main Results: CoNLL Shared Tasks	26
3.4	Discussion	29
3.4.1	Aside: Extension to Speech	33
3.5	Related Work	34
3.6	Conclusion	36
4	Noising Natural Language	37
4.1	Method	38
4.1.1	Preliminaries	38
4.1.2	Smoothing and Noising	40
4.1.3	Noising as Smoothing	41
4.1.4	Borrowing Techniques	42
4.1.5	Training and Testing	45
4.1.6	Extensions	45
4.2	Experiments	45
4.2.1	Language Modeling	45
4.2.2	Machine Translation	49
4.3	Discussion	50
4.3.1	Scaling γ via Discounting	50
4.3.2	Noised versus Unnoised Models	51
4.4	Related Work	52
4.5	Conclusion	53
5	Noising & Denoising Natural Language	54
5.1	Method	55

5.1.1	Model	55
5.1.2	Noising	57
5.1.3	Denoising	60
5.2	Experiments	60
5.2.1	CoNLL	62
5.2.2	JFLEG	63
5.3	Discussion	63
5.3.1	Realism and Human Evaluation	63
5.3.2	Noise Frequency and Diversity	65
5.3.3	Error Type Distribution Mismatch	66
5.3.4	Data Sparsity and Domain Adaptation	66
5.4	Related Work	67
5.5	Conclusion	69
6	Denoising as Style Transfer	70
6.1	Method	71
6.1.1	Noising	71
6.1.2	Denoising	73
6.2	Experiments	74
6.2.1	Data	74
6.2.2	Evaluation	75
6.2.3	Results	76
6.3	Discussion	78
6.3.1	Style and Meaning Human Evaluation	78
6.3.2	Distribution of Edits	79
6.3.3	Limitations of Noise Corpus	80
6.4	Related Work	80
6.5	Conclusion	82
7	Deployment	84
7.1	Related Work	85
7.2	Architecture	86

7.2.1 Frontend	86
7.2.2 Backend	87
7.2.3 Deep Learning Server	88
7.3 Results & Discussion	88
7.3.1 Importance of Integrations	88
7.3.2 Deep Learning-Based Suggestions Appear Useful	90
7.3.3 Diversity of Users and Writing	91
7.4 Conclusion	93
8 Conclusion	95
8.1 Other Remaining Challenges	95
8.2 Final Remarks	96
Bibliography	97

List of Tables

2.1	Example tasks that fall under the sequence transduction framework.	8
3.1	Performance on Lang-8 test set. Adding the language model results in a negligible increase in performance, illustrating the difficulty of the user-generated forum setting.	26
3.2	Development set performance. EC denotes edit classification (Section 3.2.3), and “aug” indicates data augmentation was used.	27
3.3	CoNLL 2014 test set performance. We compare to the 3 best CoNLL 2014 submissions which used combinations of MT, LM ranking, and error type-specific classifiers. We report F -score against both and single annotators, as well as each annotator scored against the other as a human ceiling. A1 and A2 denote Annotators 1 and 2.	28
3.4	Sample edits from our Lang-8 development set using only the character-based encoder-decoder network. Note that the model is able to handle misspellings (<i>Beijin</i>) as well as rare words (<i>TOEIC</i>) and emoticons (<i>∴ D</i>).	29
3.5	Sample incorrect and ambiguous edits, again using just the encoder-decoder network.	30
3.6	CoNLL development set recall for 5 most frequent error categories with and without training on data with synthesized article/determiner and noun number errors. Wci denotes wrong collocation/idiom errors.	31
3.7	Examples of the aforementioned challenging error types that our system fixed.	31

4.1	Noising schemes Example noising schemes and their bigram smoothing analogues. Here we consider the bigram probability $p(x_1, x_2) = p(x_2 x_1)p(x_1)$. Notation: $\gamma(x_{1:t})$ denotes the noising probability for a given input sequence $x_{1:t}$, $q(x)$ denotes the proposal distribution, and $N_{1+}(x, \bullet)$ denotes the number of distinct bigrams in the training set where x is the first unigram. In all but the last case we only noise the context x_1 and not the target prediction x_2 .	44
4.2	Single-model perplexity on Penn Treebank with different noising schemes. We also compare to the variational method of [37], who also train LSTM models with the same hidden dimension. Note that performing Monte Carlo dropout at test time is significantly more expensive than our approach, where test time is unchanged.	47
4.3	Perplexity on Text8 with different noising schemes.	48
4.4	Perplexities and BLEU scores for machine translation task. Results for bigram KN noising on only the source sequence and only the target sequence are given as well.	48
4.5	Perplexity of last unigram for unseen bigrams and trigrams in Penn Treebank validation set. We compare noised and unnoised models with noising probabilities chosen such that models have near-identical perplexity on full validation set.	52
5.1	Summary of training corpora.	61
5.2	Results on CoNLL 2013 (Dev) and CoNLL 2014 (Test) sets. All results use the “base” parallel corpus of 1.3M sentence pairs along with additional synthesized data (totaling 2.3M sentence pairs) except for “expanded”, which uses 3.3M nonsynthesized sentence pairs (and no synthesized data).	61
5.3	Results on the JFLEG test set (we use best hyperparameter settings from CoNLL dev set). GLEU is a variant of BLEU developed for this task; higher is better [101]. [†] Tuned to JFLEG dev set.	64

5.4	Examples of nonsynthesized and synthesized sentences from validation set. Which example (1 or 2) was synthesized? Answers: 1, 1, 2, 1, 2, 1	65
6.1	Style transfer datasets and sizes (in number of sentences). <i>Training</i> refers to examples used to synthesize noisy sentences and train the denoising model. <i>LM</i> refers to examples used to train language models for reranking and evaluation. In addition to Training and LM, 20K examples are held out for each of the dev and test sets.	75
6.2	The transfer strength for each style transfer task.	76
6.3	Meaning preservation for each style transfer task. All reported numbers scaled by 10^2 for display.	77
6.4	Fluency of each style transfer task. All reported numbers scaled by 10^3 for display.	77
6.5	Qualitative examples of style transfer results for different tasks. No parallel data outside of the initial noise corpus was used. Note that the style transfer approach can generate targets with significant syntactic changes from the source. All examples shown are without reranking during data synthesis. More qualitative examples for methods using reranking can be found at the end of this chapter.	78
6.6	Perplexities with (and without) OOVs for different datasets under seed corpus language model.	80
6.7	Additional qualitative examples of style transfer results for second two tasks (chosen since they involve more complex rephrasings). Again, no initial parallel data outside of the initial noise corpus was used. We divide the examples into those generated by models trained using (i) target style reranking and (ii) meaning preservation reranking.	83
7.1	Crio usage statistics.	91
7.2	Summary of suggestion quality for countries with different English varieties over one month period. The smallest sample size of suggestions made is 210K (UK).	93

7.3 Summary of suggestion quality for different web services over one month

period. The smallest sample size of suggestions made is 20.2K (`reddit.com`). 93

List of Figures

2.1	Figure illustrating the generic encoder-decoder model architecture we assume. Several choices are possible for the encoder and decoder architectures as well as for the attention mechanism. Here we show the outputs for a single timestep. Note that during training we may wish to <i>unroll</i> the network for multiple timesteps to parallelize some operations.	10
2.2	Expected attention matrix A when source and target are monotonically aligned (synthesized as illustrative example).	12
2.3	Toy example of beam search procedure with beam width $k = 2$. The search has been run for 3 steps and no hypothesis has terminated with $\langle \text{eos} \rangle$ yet. Edges are annotated with probabilities of tokens. Only the tokens after pruning to the top k hypotheses are shown.	15
3.1	Illustration of the encoder-decoder neural network model with two encoder hidden layers and one decoder hidden layer. Character-level reasoning allows handling of misspellings and OOVs.	20
3.2	F -score vs. length of input sentence on development set. Only bins with 10 or more sentences included.	30
3.3	Deep bi-directional recurrent neural network to map input audio features X to a distribution over output characters at each timestep t . The network contains two hidden layers with the second layer having bi-directional temporal recurrence.	34
4.1	Example training and validation curves for an unnoised model and model regularized using the bigram Kneser-Ney noising scheme.	49

4.2	Perplexity with noising on Penn Treebank while varying the value of γ_0 . Using discounting to scale γ_0 (yielding γ_{AD}) maintains gains for a range of values of noising probability, which is not true for the unscaled case.	51
4.3	Mean KL-divergence over validation set between softmax distributions of noised and unnoised models and lower order distributions. Noised model distributions are closer to the uniform and unigram frequency distributions.	51
5.1	Overview of method. We first train a noise model on a seed corpus, then apply noise during decoding to synthesize data that is in turn used to train the denoising model.	56
5.2	Model architecture used for both noising and denoising networks.	57
5.3	Illustration of random noising with beam width 2. Darker shading indicates less probable expansions. In this example, greedy decoding would yield “How are you”. Applying noise penalties, however, results in the hypotheses “How is you/he”. Note that applying a penalty does not always result in an expansion falling off the beam.	58
5.4	Percentage of time human evaluators misclassified synthesized noisy sentence Y (vs. X) when using each noising scheme, along with 95% confidence intervals. The best we can expect any scheme to do is 50%.	66
5.5	Mean edit distance between sentence pairs in X and Y after augmentation with noised sentences. <i>none</i> contains no synthesized examples while <i>clean</i> refers to the baseline of simply appending clean examples (source = target).	67
5.6	Recall vs. error type for the ten most frequent error types in our dev set. Noising improves recall uniformly across error types (See Ng et al. [103] for a description of error types).	68

6.1	An overview of our method. We first use noising to generate synthetic parallel data in both styles, then “denoise” to transfer from one style to the other. Note the parallels to the method described in Chapter 5: we use this figure to introduce the notation for the rest of this chapter as well.	72
6.2	When synthesizing noisy sentences to train the denoising model $\tilde{S} \rightarrow S$, we use style reranking to choose the noisy hypothesis h closest to the style R .	73
6.3	Normalized distributions of contiguous edit lengths (character and word-level) for NYT sentences mapped to Reddit style. We compare models trained on data generated with style reranking and meaning preservation reranking.	79
7.1	Screenshots of the final Crio interface as a browser integration.	87
7.2	Early version of Crio which was a standalone website. Users were provided with a basic text editor as well as a saved list of entries they had previously composed. Despite a similar interface to that described in Section 7.2.1, users were not retained.	89
7.3	Summary of survey question for users of early Crio system. The question was, “How would you rate the automatic grammar suggestions by [Crio] compared to other grammar checkers you have used?”	89

Chapter 1

Introduction

According to surveys of the general population [106], we average nearly an hour a day on communication. Much of this time is spent composing messages to others, whether through email, text messaging, or other channels. With the number of such messages sent and received exceeding 200 billion a day [130], tools to help improve the quality of writing and to make editing faster have the potential to improve such daily communication for billions of people.

AI writing assistants that provide spelling and grammar suggestions have existed since the 1960s [68]. More advanced tools appeared in the 1990s [61, 93], using the noisy channel model to combine the probability of certain errors with a language model probability to incorporate context. Such writing assistants are widely deployed and integrated into the web browsers, word processing software, and many other text input software that we use today. Yet, these assistants tend to be limited in both the scope and quality of their suggestions. Edits tend to be fairly local, and precision is lacking, even for common errors. To better understand how to develop assistants that are more effective, we must first understand how to improve the language models (LMs) that serve as the key underlying technology.

Following many years of count-based approaches providing the best estimates for LMs [20], recent developments in connectionist or deep learning-based approaches have become the state of the art [11, 56, 144]. However, they still have crucial limitations. An illustrative task to demonstrate their shortcomings is to sample from

such neural network-based LMs by iteratively drawing a token from the predicted next token distribution and feeding back the selected token to the LM. Here is an example from Graves [44], where a neural LM was trained on Wikipedia data and used to generate text, one character at a time:

Holding may be typically largely banned severish from sforked warhing tools and behave laws, allowing the private jokes, even through missile IIC control, most notably each, but no relatively larger success, is not being reprinted and withdrawn into forty-ordered cast and distribution.

It is immediately clear that the LM estimates are inaccurate, as the output lacks coherence, demonstrating the challenge of *data sparsity* or the *curse of dimensionality* [20]. In the LM setting where we model sequences of discrete tokens, this challenge has proven particularly difficult to surmount—despite some progress by using distributed vector representations—given the exponentially large space of possible sequences. Applied back to our motivating task of suggesting edits to writing, poor estimates result in nonrelevant or incorrect suggestions.

Within the broader challenge of data sparsity are sub-challenges such as unexpected noisy inputs or inputs that are otherwise dissimilar to what is seen during training. Unlike in most machine learning research settings, where evaluation data matches the training data distribution, AI writing assistants in the wild must deal with unexpected, possibly noisy data. Neural language models and neural sequence transduction models can be brittle black boxes, exhibiting poor and sometimes difficult to interpret behavior when given unexpected inputs [9, 90]. For sequence transduction tasks, available parallel corpora are often insufficient, and even with millions of sentence pairs, adding additional data still yields large gains in performance [25, 121].

In this thesis, we address these challenges by first better understanding the effects of noisy inputs on neural language models, then turn to developing methods for synthesizing data to help with the need for more parallel data. Throughout, we are

motivated by the problem of building an AI writing assistant that requires the ability to cope with user inputs as well as large amounts of data to perform well.¹

After some preliminary background, we examine the motivating problem of denoising text, specifically to identify and correct grammatical errors. We provide theoretical analysis of the effects of noise in neural network-based language models, leading to the development of effective schemes for synthesizing realistic noise in text. With this method, we can then synthesize data from widely available monolingual corpora. This data is used as additional training data for the tasks of suggesting grammatical and stylistic edits to text. Our approach is validated through real-world deployment and usage by thousands of users. We next outline the contributions of each chapter of this thesis in greater detail.

1.1 Contributions by Chapter

- **Chapter 2** This chapter provides background on deep learning applied to discrete sequence modeling tasks as is common for NLP tasks. This chapter covers autoregressive models, the sequence transduction problem, and common neural network models such as the encoder-decoder or sequence-to-sequence model to tackle such problems. The standard attention mechanism by which these models focus on certain portions of the source hidden states and develop “short-cut” connections between source and output is also described. After framing the problem and describing common models, this chapter also briefly covers training loss optimization as well as search-based decoding with language model rescoreing. These components form the backbone of the framework we will develop for suggesting edits to writing.
- **Chapter 3** Following the preliminaries, this chapter describes the problem that motivates the remainder of the work. Given a piece of text—usually a sentence—how can we develop models to suggest edits to that text? The purpose given in this chapter is to correct grammatical errors in the text; however,

¹Regrettably, all of the results in this thesis focus on English due to the easier availability of data resources.

later we will later consider stylistic edits as well. First, we frame the problem as a neural sequence transduction problem where, given some noisy sentence X , the objective is to output the corresponding clean sentence Y . Besides framing the problem, this chapter on denoising also applies the workhorse encoder-decoder model architecture described in Chapter 2 and details two of the standard parallel corpora used to train and evaluate models for this task (which will again appear in Chapter 5). It also describes a first attempt at a data synthesis approach to produce additional parallel data, which we will challenge in Chapter 4, and greatly improve upon in Chapter 5. To the best of our knowledge, this work is the first to apply neural network-based architectures to the problem of correcting errors or “denoising” text. It also demonstrates that a purely character-based approach is sufficient to attain good performance, despite sacrificing speed for the sake of flexibility and robustness to unexpected inputs (the out-of-vocabulary problem).

- **Chapter 4** gives the first analysis of the effects of noise specifically for neural network-based sequence models. Despite noising primitives being well developed in computer vision as well as for large-vocabulary automatic speech recognition, similar primitives have not been established for natural language processing. Naive replacements or deletions of tokens in the sentences have non-intuitive effects, as we demonstrate in this chapter, even for flexible and well-established deep learning models. After developing some theoretical analysis, we also describe noising primitives that act as effective regularizers for language modeling and machine translation. Insights from this chapter are later applied to developing data synthesis techniques for tasks towards more helpful writing assistants.
- **Chapter 5** combines the ideas developed in Chapters 3 and 4 to develop a noising and denoising system. Inspired by the findings in the previous chapter, we learn powerful neural network noise models that we then use to synthesize data for the error correction task described in Chapter 3. We find that our noising methods are able to synthesize realistic noise—validated by human evaluation

experiments—that is almost as effective as using actual noise data when training our denoising models. With powerful architectures in place as well as an effective method for synthesizing additional data, we train models that match previous state of the art on standard benchmarks.

- **Chapter 6** asks the question: Can we take inspiration from the noising and denoising method described in the previous chapter to suggest stylistic edits? Thus far we have primarily considered edits to correct errors or improve the fluency of input sentences; however, given the ability to synthesize parallel data from monolingual corpora, other types of edits such as those transferring from one language variety or genre to another seem plausible as well. We present early results demonstrating that such suggestions can sometimes be made, though unexpected changes in meaning and the open-ended nature of the task are additional challenges. This chapter presents a reranking approach as a first step towards facing these challenges.
- **Chapter 7** describes the deployed system that we developed in order to validate the usefulness of the systems we previously describe. Besides the machine learning backend that suggests edits to improve the fluency and correctness of user writing, this chapter also describes the browser integration that we built to provide suggestions across existing channels. We consider the design and human-computer interaction decisions made, as well as findings compiled from thousands of users across different roles and industries.

1.2 First Published Appearances

The majority of the contributions in this thesis with the exceptions of Chapters 6 and 7 were presented in prior work. We give here the rough correspondences between the material in each chapter and the publications in which they first appeared below:

1. Chapter 2: [138]
2. Chapter 3: [139], [91]

3. Chapter 4: [140]

4. Chapter 5: [141]

Chapter 2

Background

This chapter attempts to provide the background knowledge required for the remaining chapters in the thesis. Before discussing the problems specific to writing assistants, we first consider the broader problem of tasks that involve generating text, conditioned on some input information.

2.1 Setting

We consider modeling discrete sequences of text tokens. Given a sequence $U = (u_1, u_2, \dots, u_S)$ over the vocabulary V , we seek to model

$$p(U) = \prod_{t=1}^S p(u_t | u_{<t}) \tag{2.1}$$

where $u_{<t}$ denotes u_1, u_2, \dots, u_{t-1} , and equality follows from the chain rule of probability. Depending on how we choose to tokenize the text, the vocabulary can contain the set of characters, word-pieces/byte-pairs, words, or some other unit. For the tasks we consider in this chapter, we divide the sequence U into an input or *source* sequence X (that is always provided in full) and an output or *target* sequence Y . For example, X might be a sentence written in a casual style and Y the sentence written

Task	X (example)	Y (example)
language modeling	none (empty sequence)	tokens from news corpus
machine translation	source sequence in English	target sequence in French
grammar correction	noisy, ungrammatical sentence	corrected sentence
summarization	body of news article	headline of article
dialogue	conversation history	next response in turn
<i>Related tasks under the sequence transduction framework.</i>		
speech transcription	audio / speech features	text transcript
image captioning	image	caption describing image
question answering	supporting text + KB + Q	answer

Table 2.1: Example tasks that fall under the sequence transduction framework.

in a formal style. In this case, we model

$$p(Y|X) = \prod_{t=1}^T p(y_t|X, y_{<t}) \quad (2.2)$$

Note that this is a generalization of (2.1); we consider $p(Y|X)$ from here on. This formalism encompasses many tasks in natural language processing—see Table 2.1 for more examples of sequence transduction tasks.

Aside: Beyond the tasks described in the first half of Table 2.1, many of the techniques described here also extend to tasks at the intersection of text and other modalities. For example, in speech recognition, X may be a sequence of features computed on short snippets of audio, with Y being the corresponding text transcript, and in image captioning X is an image (which is not so straightforward to express as a sequence) and Y the corresponding text description. Following is a summary of the notation we use in this chapter.

Summary of Notation

Symbol	Shape	Description
V	scalar	size of output vocabulary
X	S	input/source sequence (x_1, \dots, x_S) of length S
Y	T	output/target sequence (y_1, \dots, y_T) of length T
E	(S, h)	encoder hidden states where E_j denotes representation at timestep j
D	(T, h)	decoder hidden states where D_i denotes representation at timestep i
A	(T, S)	attention matrix where A_{ij} is attention weight at i th decoder timestep on j th encoder state representation
H	<i>varies</i>	hypothesis in hypothesis set \mathcal{H} during decoding
$s(H)$	scalar	score of hypothesis H during beam search

Minibatch size dimension is omitted from the shape column.

2.2 Neural Sequence Transduction Models

Before neural network-based approaches, count-based methods [20] and methods involving learning phrase pair probabilities were used for language modeling and sequence transduction. Since then, neural sequence transduction models, also referred to as encoder-decoder models or sequence-to-sequence models, have rapidly exceeded the performance of prior systems despite having comparatively simple architectures, trained end-to-end to map source directly to target.

Several such architectures have demonstrated strong results.

- *Recurrent neural networks* (RNNs) use shared parameter matrices across different time steps and combine the input at the current time step with the previous hidden state summarizing all previous time steps [44, 95, 128, 22]. Different gating mechanisms have been developed to try and ease optimization [51, 22].

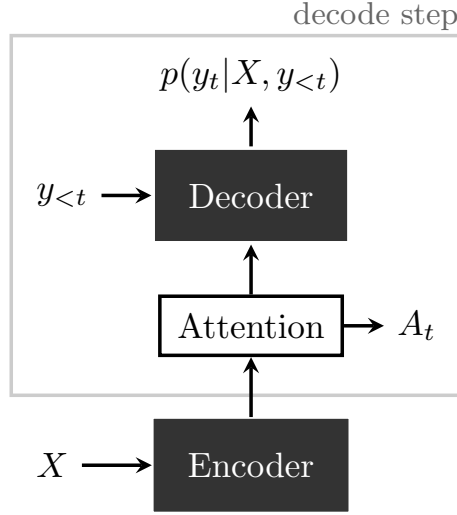


Figure 2.1: Figure illustrating the generic encoder-decoder model architecture we assume. Several choices are possible for the encoder and decoder architectures as well as for the attention mechanism. Here we show the outputs for a single timestep. Note that during training we may wish to *unroll* the network for multiple timesteps to parallelize some operations.

- *Convolutional neural networks* (CNNs). Convolutions with kernels reused across timesteps can also be used with masking to avoid peeking ahead at future inputs during training (see Section 2.4 for an overview of the training procedure) [60, 40]. Using convolutions has the benefit during training of parallelizing across the time dimension instead of computing the next hidden state one step at a time.
- Both recurrent and convolutional networks for modeling sequences typically rely on a per timestep *attention* mechanism [6] that acts as a shortcut connection between the target output prediction and the relevant source input hidden states. At a high-level, at decoder timestep i , the decoder representation D_i is used to compute a weight α_{ij} for each encoder representation E_j . For example, this could be done by using the dot product $D_i^\top E_j$ as the logits before applying the

softmax function. Hence

$$\alpha_{ij} = \exp(D_i^\top E_j) / \sum_{k=1}^S \exp(D_i^\top E_k)$$

The weighted representation $\sum_{j=1}^S \alpha_{ij} E_j$ is then fed into the decoder along with X and $y_{<i}$. More recent models which rely purely on attention mechanisms with masking have also been shown to obtain as good or better results as RNN or CNN-based models [131]. We describe the attention mechanism in more detail in Section 2.3.

In general, for recent encoder-decoder models the following two conditions hold:

1. The model performs next-step prediction of the next target conditioned on the source and previous targets, i.e. it models $p(y_t | X, y_{<t})$.
2. The model uses an attention mechanism (resulting in an attention matrix A), which eases training, is simple to implement and reasonably efficient to compute in most cases, and has become a standard component of encoder-decoder models.

Figure 2.1 illustrates the backbone architecture that is used throughout this thesis.

2.3 Attention

The basic attention mechanism used to “attend” to portions of the encoder hidden states during each decoder timestep has many extensions and applications. Attention can also be over previous decoder hidden states, in what is called self-attention [131], or used over components separate from the encoder-decoder model instead of over the encoder hidden states [42]. It can also be used to monitor training progress (by inspecting whether a clear alignment develops between encoder and decoder hidden states), as well as to inspect the correspondences between the input and output sequence that the network learns.

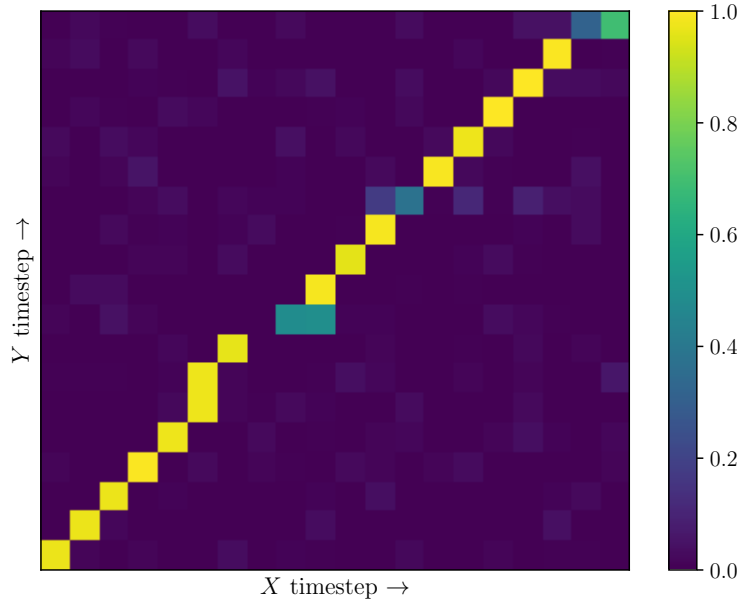


Figure 2.2: Expected attention matrix A when source and target are monotonically aligned (synthesized as illustrative example).

The attention matrix A has T rows and S columns, where T is the number of output timesteps and S is the number of input timesteps. Every row A_i is a discrete probability distribution over the encoder hidden states, which in this guide we assume to be equal to the number of input timesteps S . In the case that an encoder column $A_{\cdot j}$ has no entry over some threshold (say 0.5), this suggests that the corresponding encoder input was ignored by the decoder. Likewise, if $A_{\cdot j}$ has multiple values over threshold, this suggests those encoder hidden states were repeatedly used during multiple decoder timesteps. Figure 2.2 shows an attention matrix we might expect for a well-trained network where source and target are well-aligned (e.g. English→French translation). For a great overview and visualizations of attention and RNN models, also see [107].

2.4 Training Overview

During training, we optimize over the model parameters θ the sequence cross-entropy loss

$$\ell(\theta) = - \sum_{t=1}^T \log p(y_t | X, y_{<t}; \theta). \quad (2.3)$$

thus maximizing the log-likelihood of the training data. Previous ground truth inputs are given to the model when predicting the next index in the sequence, a training method sometimes referred to (unfortunately) as *teacher forcing*. Due to the inability to fit current datasets into memory as well as for faster convergence, gradient updates are computed on minibatches of training sentences. *Stochastic gradient descent* (SGD) as well as optimizers such as Adam [62] have been shown to work well empirically.

Recent research has also explored other methods for training sequence models, such as by using reinforcement learning or a separate adversarial loss [41, 79, 80, 7, 3]. As of this writing, however, the aforementioned training method is the primary workhorse for training such models.

2.5 Decoding Overview

During decoding, we are given the source sequence X and seek to generate the target \hat{Y} that maximizes some scoring function $s(\hat{Y})$.¹ In *greedy decoding*, we simply take the argmax over the softmax output distribution for each timestep, then feed that as the input for the next timestep. Thus at any timestep we only have a single hypothesis. Although greedy decoding can work surprisingly well, note that it often does *not* result in the most probable output hypothesis, since there may be a path that is more probable overall despite including an output which was not the argmax (this also holds true for most scoring functions we may choose).

Since it's usually intractable to consider every possible \hat{Y} due to the branching factor and number of timesteps, we instead perform *beam search*, where we iteratively expand each hypotheses one token at a time, and at the end of every search iteration

¹The scoring function may also take X or other tensors as input, but for simplicity we consider just \hat{Y} .

we only keep the k best (in terms of $s(\cdot)$) hypotheses, where k is the *beam width* or *beam size*. Here’s the full beam search procedure, in more detail:

1. We begin the beam procedure with the start-of-sequence token, $\langle \text{sos} \rangle$. Thus our set of hypothesis $\mathcal{H} = \{[\langle \text{sos} \rangle]\}$ consisting of the single hypothesis $H = [\langle \text{sos} \rangle]$, a list with only the start token.
2. Repeat, for $t = 1, 2, \dots, T_{\max}$:
 - (a) Repeat, for $H \in \mathcal{H}$:
 - i. Repeat, for every u in the vocabulary V with probability $p_{ut} = p(u|X, H)$:
 - A. Add the hypothesis $H_{\text{new}} = [\langle \text{sos} \rangle, h_1, \dots, h_{t-1}, u]$ to \mathcal{H} .
 - B. Compute and cache $s(H_{\text{new}})$. For example, if $s(\cdot)$ simply computes the cumulative log-probability of a hypothesis, we have $s(H_{\text{new}}) = s(H) + \log p_{ut}$.
 - (b) If any of the hypotheses end with the end-of-sequence token $\langle \text{eos} \rangle$, move that hypothesis to the list of terminated hypotheses $\mathcal{H}_{\text{final}}$.
 - (c) Keep the k best remaining hypotheses in \mathcal{H} according to $s(\cdot)$.²
3. Finally, we return $H^* = \arg \max_{H \in \mathcal{H}_{\text{final}}} s(H)$. Since we are now considering completed hypotheses, we may also wish to use a modified scoring function $s_{\text{final}}(\cdot)$.

One surprising result with neural models is that relatively small beam sizes yield good results with rapidly diminishing returns. Further, larger beam sizes can even yield (slightly) worse results. For example, a beam size of 8 may only work marginally better than a beam size of 4, and a beam size of 16 may work worse than 8 [65].

Finally, oftentimes incorporating a language model in the scoring function can help improve performance. Since LMs only need to be trained on the target corpus, we can train language models on much larger corpuses that do not have parallel data.

²Likewise, we can also store other information for each hypothesis, such as A_t .

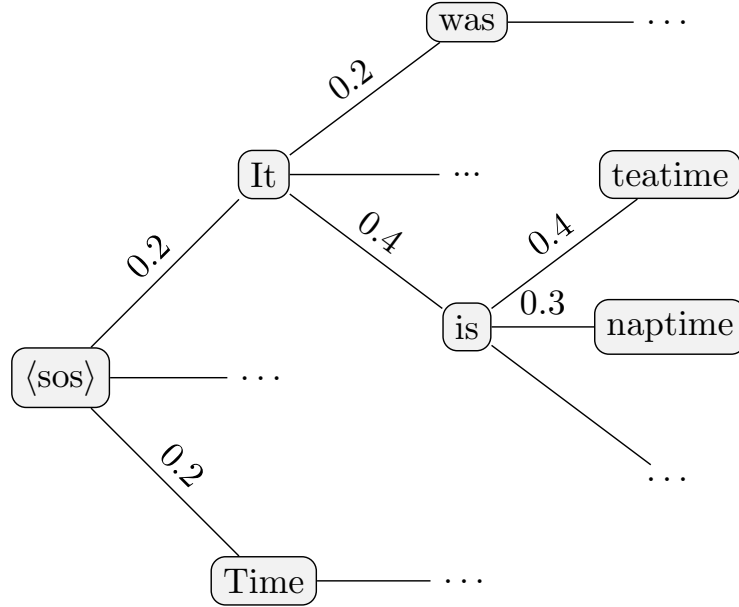


Figure 2.3: Toy example of beam search procedure with beam width $k = 2$. The search has been run for 3 steps and no hypothesis has terminated with $\langle \text{eos} \rangle$ yet. Edges are annotated with probabilities of tokens. Only the tokens after pruning to the top k hypotheses are shown.

Our objective in decoding is to maximize the joint probability:

$$\arg \max_{\hat{Y}} p(X, \hat{Y}) = \arg \max_{\hat{Y}} p(X|\hat{Y})p(\hat{Y}) \quad (2.4)$$

However, since we are given X and not Y , it's intractable to maximize over $p(X|\hat{Y})$. In practice, we instead maximize the pseudo-objective

$$\arg \max_{\hat{Y}} p(\hat{Y}|X)p(\hat{Y}).$$

If our original score is $s(H)$, which we assume involves a $\log p(H|X)$ term, then the LM-augmented score is

$$s(H) + \lambda \log p_{\text{LM}}(H)$$

where λ is a hyperparameter to balance the LM and decoder scores. In practice, this simple procedure works fairly well.

2.6 Evaluation

One of the key challenges for developing language modeling or sequence transduction systems is that there is often no satisfying automated metric for evaluating the final output of the system. Unlike classification and sequence labeling tasks, we (as of now) cannot precisely measure the output quality of many such systems barring human evaluation. Perplexity does not always correlate well with downstream metrics [21], automated or otherwise. Common metrics based on n -gram overlap such as ROUGE [82] and BLEU [108] are only rough approximations, and often do not capture linguistic fluency and coherence [27, 85]. Such metrics are especially problematic in more open-ended generation tasks. For the task considered in this thesis of suggesting edits, the typical metric used is F -score based on gold edits. However, oftentimes there are multiple valid edits, some of which have not been annotated. Beyond just correctness, edits involving fluency and style are more difficult to evaluate, and often labels are not available.

All that said, recent results have shown that though automated metrics are not great at distinguishing between systems once performance passes some baseline, they nonetheless are useful for finding examples where performance is poor, and can also be consistent for evaluating similar systems [105]. Thus, despite issues with current automated evaluation metrics, we often use them nonetheless in this thesis.

2.7 Other Topics

Due to some of the current limitations of neural language models and sequence transduction models, topics we did not cover in this chapter include:

- Natural language understanding and semantics. While impressive work has been done in learning word embeddings [97, 110], the goal of learning “thought vectors” for sentences has remained more elusive [64].
- Sequence labeling or classification tasks. Although we do not cover these explicitly, most common methods for sequence classification are a simple and natural extension of the methods described in this chapter.

- How to capture long-term dependencies (beyond a brief discussion of attention) or maintain global coherence. This remains a challenge due to the curse of dimensionality as well as neural networks failing to learn more abstract concepts from the predominant next-step prediction training objective.
- How to interface models with a knowledge base, or other structured data that cannot be supplied in a short piece of text. Some recent work has used pointer mechanisms towards this end [133].

2.8 Conclusion

In this chapter we described the language modeling and sequence transduction tasks that underly the techniques in the remaining chapters, the current neural models that have achieved state-of-the-art performance, and the standard training and decoding procedures. With these pieces in place, we return to the goal of developing more useful AI writing assistants, starting with the task of suggesting edits to correct errors in writing.

Chapter 3

Denoising Natural Language

Grammar is a piano I play by ear.

Joan Didion

To work towards building an AI writing assistant, in this chapter we formalize the task of suggesting edits to text. Systems that provide such writing feedback have great potential to assist second-language learners [74] as well as native writers in their communication, such as with emails, SMS messages [5], and longer documents [13]. Although tools such as spell checkers are useful and widely used [68], detecting and fixing more general errors in natural language, even at the sentence level, remains far from solved [103, 18].

We focus on systems that can *both* detect errors as well as make edit suggestions simultaneously, in contrast to earlier work that first focuses solely on detecting errors [23]. Early examples of such work include spelling correction systems that use the noisy channel model [61, 93]. Beyond spelling, subsequent research focuses on training classifiers for a small number of error types, such as article or preposition errors [45, 114] or verb form errors [75] for which the edits are then easily determined. More recent methods then consider a broader range of error classes by leveraging language models to score n -grams or by using statistical machine translation approaches [103, 16, 74] based on newly available parallel learner corpora [99]. Such translation-based approaches produce correction hypotheses using beam search [28],

often with rescoring as described in Chapter 2. Translation-based approaches prior to the work in this chapter, however, did not use distributed representations [11] to more efficiently handle orthographic errors in spelling, capitalization, and punctuation.

As a motivating example, consider the following incorrect sentence: “*I visitted Tokyo on Nov 2003. :)*”. Several errors in this sentence illustrate the difficulties in the error correction setting. First, the sentence contains a misspelling, *visitted*, an issue for systems with fixed vocabularies. Second, the sentence contains rare words such as *2003* as well as punctuation forming an emoticon *:)*, issues that may require special handling. Finally, the use of the preposition *on* instead of *in* when not referring to a specific day is non-idiomatic, demonstrating the complex patterns that must be captured to suggest good corrections. In hopes of capturing such complex phenomena, we use a neural network-based method.

Building on recent work in language modeling and machine translation, we propose an approach to grammatical error correction based on an encoder-decoder recurrent neural network trained on a parallel corpus containing “good” and “bad” sentences (Figure 3.1). When combined with a language model, our system outperforms previous results on the CoNLL 2014 Shared Task, beating systems using statistical machine translation systems, rule-based methods, and task-specific features. Our system naturally handles orthographic errors and rare words, and can flexibly correct a variety of error types. We further find that augmenting the network training data with sentences containing synthesized errors can result in significant gains in performance, though with a surprising catch.

3.1 Model Architecture

As described in Chapter 2, we use an encoder-decoder model [127, 22]. The encoder maps the input sentence to a higher-level representation with a pyramidal bi-directional RNN architecture similar to that of Chan et al. [19]. The decoder is also a recurrent neural network that uses a content-based attention mechanism [6] to attend to the encoded representation and generate the output sentence one character at a time.

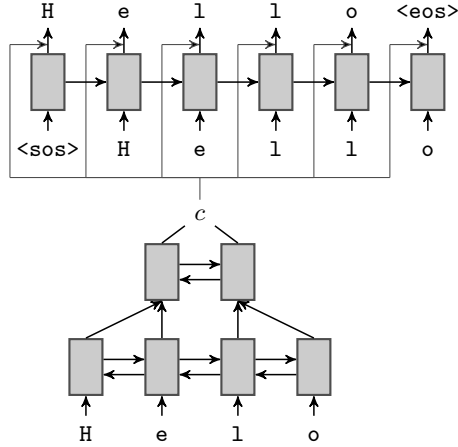


Figure 3.1: Illustration of the encoder-decoder neural network model with two encoder hidden layers and one decoder hidden layer. Character-level reasoning allows handling of misspellings and OOVs.

3.1.1 Character-Level Reasoning

Our neural network model operates at the character level, in the encoder as well as the decoder. This is for two reasons, as illustrated by our motivating example. First, we do not assume that the inputs are spell-checked and often find spelling errors in the sentences written by English learners in the datasets we consider. Second, word-level neural MT models with a fixed vocabulary are poorly suited to handle OOVs such as multi-digit numbers, emoticons, and web addresses [44], though recent work has proposed workarounds for this problem [89].¹ Despite longer sequences in the character-based model, optimization does not seem to be a significant issue, since the network often only needs to copy characters from source to target.

¹Since the work in this chapter, subword tokenization schemes such as byte-pair encoding [120] have also become an alternative way to handle this issue.

3.1.2 Encoder Network

Given the input vector x_t , the forward, backward, and combined activations of the j th hidden layer are computed as:

$$\begin{aligned} f_t^{(j)} &= \text{GRU}(f_{t-1}^{(j)}, c_t^{(j-1)}), \\ b_t^{(j)} &= \text{GRU}(b_{t+1}^{(j)}, c_t^{(j-1)}), \\ h_t^{(j)} &= f_t^{(j)} + b_t^{(j)} \end{aligned}$$

where GRU denotes the gated recurrent unit function, which, similar to long short-term memory units (LSTMs), have shown to improve the performance of RNNs [22, 51].

The input from the previous layer input $c_t^{(0)} = x_t$ and

$$c_t^{(j)} = \tanh \left(W_{\text{pyr}}^{(j)} \left[h_{2t}^{(j-1)}, h_{2t+1}^{(j-1)} \right]^\top + b_{\text{pyr}}^{(j)} \right)$$

for $j > 0$. The weight matrix W_{pyr} thus reduces the number of hidden states for each additional hidden layer by half, and hence the encoder has a pyramid structure. At the final hidden layer we obtain the encoded representation c consisting of $\lceil T/2^{N-1} \rceil$ hidden states, where N denotes the number of hidden layers.

3.1.3 Decoder Network

The decoder network is recurrent neural network using gated recurrent units with M hidden layers. After the final hidden layer the network also conditions on the encoded representation c using an attention mechanism.

At the j th decoder layer the hidden activations are computed as

$$d_t^{(j)} = \text{GRU}(d_{t-1}^{(j)}, d_t^{(j-1)}),$$

with the output of the final hidden layer $d_t^{(M)}$ then being used as part of the content-based attention mechanism similar to that proposed by Bahdanau et al. [6]:

$$\begin{aligned} u_{tk} &= \phi_1(d_t^{(M)})^\top \phi_2(c_k) \\ \alpha_{tk} &= \frac{\exp(u_{tk})}{\sum_j \exp(u_{tj})} \\ a_t &= \sum_j \alpha_{tj} c_j \end{aligned}$$

where ϕ_1 and ϕ_2 represent feedforward affine transforms followed by a tanh nonlinearity. The weighted sum of the encoded hidden states a_t is then concatenated with $d_t^{(M)}$, and passed through another affine transform followed by a ReLU nonlinearity before the final softmax output layer.

The loss function is the cross-entropy loss per time step summed over the output sequence y , as described in the previous chapter. Figure 3.1 illustrates the model architecture.

3.1.4 Pyramid Structure

Although character-level models reduce the softmax over the vocabulary at each time step over word-level models, they also increase the total number of time-steps of the RNN. The content-based attention mechanism must then consider all the encoder hidden states $c_{1:T}$ at every step of the decoder. Thus we use a pyramid architecture, which reduces computational complexity (as shown by Chan et al. [19]). For longer batches, we observe over a $2\times$ speedup for the same number of parameters when using a 400 hidden unit per layer model with 3 hidden layers ($4\times$ reduction of steps in c).

3.2 Decoding

While it is simpler to integrate a language model by using it as a re-ranker, here the language model probabilities are combined with the encoder-decoder network through beam search. This is possible because the attention mechanism in the decoder network

prevents the decoded output from straying too far from the source sentence.

3.2.1 Language Model

To model the distribution

$$P_{\text{LM}}(y_{1:T}) = \prod_{t=1}^T P(y_t|y_{<t}) \quad (3.1)$$

we build a Kneser-Ney smoothed 5-gram language model on a subset of the Common Crawl Repository² collected during 2012 and 2013. After pruning, we obtain 2.2 billion n -grams. To build and query the model, we use the KenLM toolkit [48].

3.2.2 Beam Search

For inference we use a beam search decoder combining the neural network and the language model likelihood. Similar to the method described in Chapter 2, at step k , we rank the hypotheses on the beam using the score

$$s_k(y_{1:k}|x) = \log P_{\text{NN}}(y_{1:k}|x) + \lambda \log P_{\text{LM}}(y_{1:k})$$

where the hyper-parameter λ determines how much the language model is weighted. To avoid penalizing longer hypotheses, we additionally normalize scores by the number of words in the hypothesis $|y|$. Since decoding is done at the character level, the language model probability $P_{\text{LM}}(\cdot)$ is only incorporated after a space or end-of-sentence symbol is encountered.

3.2.3 Controlling Precision

For many error correction tasks, precision is emphasized more than recall; for users, an incorrect suggestion is worse than a missed mistake.

In order to filter spurious edits, we train an edit classifier to classify edits as correct or not. We run our decoder on uncorrected sentences from our training data

²<http://commoncrawl.org>

to generate candidate corrected sentences. We then align the candidate sentences to the uncorrected sentences by minimizing the word-level Levenshtein distance between each candidate and uncorrected sentence. Contiguous segments that do not match are extracted as proposed edits³. We repeat this alignment and edit extraction process for the gold corrected sentences and the uncorrected sentences to obtain the gold edits. “Good” edits are defined as the intersection of the proposed and gold edits and “bad” edits are defined as the proposed edits not contained in the gold edits. We compute edit features and train a multilayer perceptron binary classifier on the extracted edits to predict the probability of an edit being correct. The features computed on an edit $s \rightarrow t$ are:

- **edit distance features:** normalized word and character lengths of s and t , normalized word and character insertions, deletions, and substitutions between s and t .
- **embedding features:** sum of 100 dimensional GloVe [110] vectors of words in s and t , GloVe vectors of left and right context words in s .

In order to filter incorrect edits, we only accept edits whose predicted probability exceeds a threshold p_{\min} . This assumes that classifier probabilities are reasonably calibrated [104]. Edit classification improves precision with a small drop in recall; most importantly, it helps filter edits where the decoder network misbehaves and t deviates wildly from s .

3.3 Experiments

We perform experiments using two datasets of corrected sentences written by English learners. The first is the Lang-8 Corpus, which contains erroneous sentences and their corrected versions collected from a social language learner forum [129]. Due to the online user-generated setting, the Lang-8 data is noisy, with sentences often containing misspellings, emoticons, and other loose punctuation. Sample sentences are shown in Table 3.4.

³Note this is an approximation and cannot distinguish side-by-side edits as separate edits.

The other dataset we consider comes from the CoNLL 2013 and 2014 Shared Tasks, which contain about 60K sentences from essays written by English learners with corrections and error type annotations. We use the larger Lang-8 Corpus primarily to train our network, then evaluate on the CoNLL Shared Tasks.

3.3.1 Training and Decoding Details

Our pyramidal encoder has 3 layers, resulting in a factor 4 reduction in the sequence length at its output, and our decoder RNN has 3 layers as well. Both the encoder and decoder use a hidden size of 400 and gated recurrent units (GRUs), which along with LSTMs [50] have been shown to be easier to optimize and preserve information over many time steps better than vanilla recurrent networks.

Our vocabulary includes 98 characters: the printing ASCII character set and special $\langle \text{sos} \rangle$, $\langle \text{eos} \rangle$, and $\langle \text{unk} \rangle$ symbols indicating the start-of-sentence, end-of-sentence, and unknown symbols, respectively.

To train the encoder-decoder network we use the Adam optimizer [62] with a learning rate of 0.0003, default decay rates β_1 and β_2 , and a minibatch size of 128. We train for up to 40 epochs, selecting the model with the lowest perplexity on the Lang-8 development set. We found that using dropout [125] at a rate of 0.15 on the non-recurrent connections [111] helped reduce perplexity. We use uniform initialization of the weight matrices in the range $[-0.1, 0.1]$ and zero initialization of biases.

Decoding parameter λ and edit classifier threshold p_{\min} were chosen to maximize performance on the development sets of the datasets described. All results were obtained using a beam width of 64, which seemed to provide a good trade-off between speed and performance.

3.3.2 Noisy Data: Lang-8 Corpus

We use the train-test split provided by the Lang-8 Corpus of Learner English [129], which contains 100K and 1K entries with about 550K and 5K parallel sentences, respectively. We also split 5K sentences from the training set to use as a separate

Method	Test BLEU
No edits	59.54
Spell check	58.91
RNN	61.63
RNN + LM	61.70

Table 3.1: Performance on Lang-8 test set. Adding the language model results in a negligible increase in performance, illustrating the difficulty of the user-generated forum setting.

development set for model and parameter selection.

Since we do not have gold annotations that distinguish side-by-side edits as separate edits, we report BLEU score⁴ using just the encoder-decoder network as well as when combined with the n -gram language model (Table 3.1). Note that since there may be multiple ways to correct an error and some errors are left uncorrected, the baseline of using uncorrected sentences is more difficult to improve upon than it may initially appear. As another baseline we apply the top suggestions from a spell checker with default configurations⁵. We suspect due to proper nouns, acronyms, and inconsistent capitalization conventions in Lang-8, however, this actually decreased BLEU slightly. To the best of our knowledge, no other work has reported results on this challenging task.

3.3.3 Main Results: CoNLL Shared Tasks

Description For our second set of experiments we evaluate on the CoNLL 2014 Shared Task on Grammatical Error Correction [102, 103]. We use the revised CoNLL 2013 test data with all error types as a development set for parameter and model selection with the 2014 test data as our test set. The 2013 test data contains 1381 sentences with 3470 errors in total, and the 2014 test data contains 1312 sentences with 3331 errors. The CoNLL 2014 training set contains 57K sentences with the

⁴Using case-sensitive `multi-bleu.perl` from Moses.

⁵Hunspell v1.3.4, <https://hunspell.github.io>

Method	P	R	$F_{0.5}$
RNN	42.96	6.27	19.81
RNN aug	49.30	10.10	27.75
RNN + LM	43.27	15.14	31.55
RNN aug + LM	46.94	17.11	34.81
RNN aug + LM + EC	51.38	15.83	35.45

Table 3.2: Development set performance. EC denotes edit classification (Section 3.2.3), and “aug” indicates data augmentation was used.

corresponding gold edits by a single annotator. The 2013 test set is also only labeled by a single annotator, while the 2014 test set has two separate annotators.

We use the NUS MaxMatch scorer [29] v3.2 in order to compute the precision (P), recall (R), and F -score for our corrected sentences. Since precision is considered more important than recall for the error correction task, $F_{0.5}$ score is reported as in the CoNLL 2014 Challenge. We compare to the top submissions in the 2014 Challenge as well as the method by Susanto [126], which combines 3 of the weaker systems to achieve a previous state-of-the-art result. All results reported on the 2014 test set exclude alternative corrections submitted by the participants.

Synthesizing Errors In addition to the Lang-8 training data, we include the CoNLL 2014 training data in order to train the encoder-decoder network. Following prior work, we additionally explore synthesizing additional sentences containing errors using the CoNLL 2014 training data [35, 116]. Our data augmentation procedure generates synthetic errors for two of the most common error types in the development set: article or determiner errors (ArtOrDet) and noun number errors (Nn). Similar to Felice and Yuan [35], we first collect error distribution statistics from the CoNLL 2014 training data. For ArtOrDet errors, we estimate the probability that an article or determiner is deleted, replaced with another determiner, or inserted before the start of a noun phrase. For Nn errors, we estimate the probability that it is replaced with its singular or plural form. To obtain sentence parses we use the Stanford CoreNLP Toolkit [92]. Example synthesized errors:

Method	P	R	$F_{0.5}$
AMU	41.62	21.40	35.01
CUUI	41.78	24.88	36.79
CAMB	39.71	30.10	37.33
Susanto [126]	53.55	19.14	39.39
Ours (no EC)	45.86	26.40	39.97
Ours (+ EC)	49.24	23.77	40.56
Ours (A1)	32.56	14.76	26.23
Ours (A2)	44.04	14.83	31.59
A1 (A2)	50.47	32.29	45.36
A2 (A1)	37.14	45.38	38.54

Table 3.3: CoNLL 2014 test set performance. We compare to the 3 best CoNLL 2014 submissions which used combinations of MT, LM ranking, and error type-specific classifiers. We report F -score against both and single annotators, as well as each annotator scored against the other as a human ceiling. A1 and A2 denote Annotators 1 and 2.

- **ArtOrDet**: They will generate and brainstorm *the* innovative ideas.
- **Nn**: Identification is becoming more important in our *society* \rightarrow *societies*.

Errors are introduced independently according to their estimated probabilities by iterating over the words in the training sentences, and we produce two corrupted versions of each training sentence whenever possible. The original Lang-8 training data contains 550K sentence pairs. Adding the CoNLL 2014 training data results in about 610K sentence pairs, and after data augmentation we obtain a total of 720K sentence pairs. We examine the benefits of synthesizing errors in Section 3.4.

Results Results for the development set are shown in Table 3.2, and results for the CoNLL 2014 test set in Table 3.3. On the CoNLL 2014 test set, which contains the full set of 28 error types, our method out-performs all systems from the 2014 Challenge as well as a system combination method [126]. Methods from the 2014 Challenge used statistical machine translation, language model ranking, rule-based approaches, and error type-specific features and classifiers, often in combination. System descriptions

Original	Proposed
1 It 's <u>heavy rain</u> today	It 's <u>raining heavily</u> today
2 Everyone wants to be <u>success</u> .	Everyone wants to be <u>successful</u> .
3 <u>On the</u> 3 weeks , I learned many things .	<u>In the last</u> 3 weeks , I learned many things .
4 <u>this is the</u> first entry ! : D	<u>This is my</u> first entry ! : D
5 Help me <u>getting English skill</u> , please .	Help me <u>improve my English skills</u> , please .
6 <u>At</u> last night , the 24th of June 2010 was big night for the Japanese national team and heaps of fans .	Last night , the 24th of June 2010 was <u>a</u> big night for the Japanese national team and heaps of fans .
7 I <u>start to learning</u> English again .	I <u>am starting to learn</u> English again .
8 I went to <u>Beijin</u> in China <u>for</u> four days <u>in</u> this week .	I went to <u>Beijing</u> in China four days this week .
9 After a long day , <u>I and my roommate</u> usually sit down , drink coffee and listen to music .	After a long day , <u>my roommate and I</u> usually sit down , drink coffee and listen to music .
10 Do you know <u>a Toeic</u> ?	Do you know about <u>TOEIC</u> ?

Table 3.4: Sample edits from our Lang-8 development set using only the character-based encoder-decoder network. Note that the model is able to handle misspellings (*Beijin*) as well as rare words (*TOEIC*) and emoticons (: *D*).

for participating teams are given in Ng et al. [103].

3.4 Discussion

Qualitative Analysis We present examples of correct and incorrect edits on Lang-8 development set in Table 3.4 and Table 3.5. Despite operating at the character level, the network is occasionally able to perform rearrangements of words to form common phrases (e.g. *I and my roommate* → *my roommate and I*) and insert and delete words where appropriate. On the other hand, the network can also sometimes mangle rare words (*Moodysson* → *Moodysnott*) and fail to split common words missing a separating space (*withthe* → *withth*), suggesting that while common patterns are captured, the

	Original	Proposed
1	Broke my heart	<u>I</u> broke my heart
2	I want to big size bag	I want to <u>be</u> a big size bag
3	This is typical Japanese male <u>hobit</u>	This is <u>a</u> typical Japanese male <u>hobby</u>
4	I 'm so sorry to miss Lukas <u>Moodysson</u> 's Lijia 4-ever .	I 'm so sorry to miss Lukas <u>Moodyssnot</u> Lijia 4-ever .
5	The match is the Rockets <u>withthe</u> Bulls .	The match is the Rockets <u>withth</u> Bulls .

Table 3.5: Sample incorrect and ambiguous edits, again using just the encoder-decoder network.

network lacks semantic understanding.

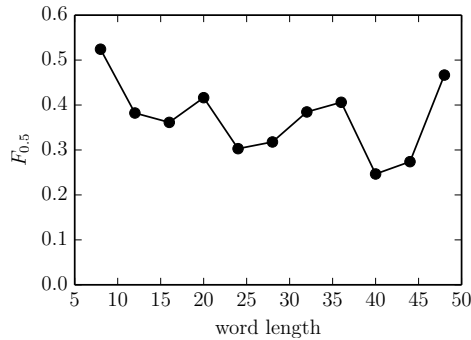


Figure 3.2: F -score vs. length of input sentence on development set. Only bins with 10 or more sentences included.

Performance Breakdown While the encoder-decoder network can itself produce modifications, on less noisy datasets such as the CoNLL Challenge datasets a language model can greatly improve performance. Increasing the language model weight λ tends to improve recall at the expense of precision. On the other hand, using edit classification to filter spurious edits increases precision, often with smaller drops in recall. We do not observe a trend of decreasing F -score for a wide range of sentence lengths (Figure 3.2), likely due to the attention mechanism, which helps to prevent the decoded output from diverging from the input sentence.

We report the inter-annotator agreement in Table 3.3, which gives a possible bound on the F -score for this task.

Type	Count	R no aug	R aug
ArtOrDet	717	20.08	29.14
Wci	434	2.30	1.61
Nn	400	31.50	51.00
Preposition	315	13.01	7.93
Word form	223	26.90	19.73

Table 3.6: CoNLL development set recall for 5 most frequent error categories with and without training on data with synthesized article/determiner and noun number errors. Wci denotes wrong collocation/idiom errors.

Type	Description	Original	Proposed
Mec	Spelling, punctuation, capitalization, etc.	Another identification is <u>Implanting</u> RFID chips ...	Another identification is <u>implanting</u> RFID chips ...
Rloc-	Redundancy	...it seems that our freedom <u>of doing things</u> is being invaded.	...it seems that our freedom is being invaded.
Wci	Wrong collocation/idiom	Every coin has <u>its</u> two sides.	Every coin has two sides.

Table 3.7: Examples of the aforementioned challenging error types that our system fixed.

Effects of Data Augmentation We obtain promising improvements using data augmentation, boosting $F_{0.5}$ -score on the development set from 31.55 to 34.81. For the two error types where we synthesize data (article or determiner and noun number) we observe significant increases in recall, as shown in Table 3.6. The same phenomenon has been observed by Rozovskaya et al. [116]. Interestingly, the recall of other error types (see Ng et al. [103] for descriptions) decreases. This actually results in performance on the test set decreasing slightly (0.1 F -score). We surmise this is because the additional training data contains only ArtOrDet and Nn errors, and hence the network is encouraged to simply copy the output when those error types are not present. In Chapter 4 we will try and better understand the effects of simple noise schemes, and in Chapter 5 develop a method to synthesize diverse errors that consistently improves performance.

Challenging Error Types We now examine a few illustrative error types from

the CoNLL Challenges that originally motivated our approach: orthographic (Mec), redundancy (Rloc-), and idiomatic errors (Wci). Since the 2013 Challenge did not score these error types, we compare our recall to those of participants in the 2014 Challenge [103].⁶ Note that systems only predict corrected sentences and not error types, and hence precision is not compared. We use the results from our final system, including both data augmentation and edit classification. Some examples of these error types are shown in Table 3.7.

- **Mec:** We obtain a recall of 37.17 on the Mec error type, higher than all the 2014 Challenge teams besides one team (RAC) that used rule-based methods to attain 43.51 recall. The word/phrase-based translation and language modeling approaches do not seem to perform as well for fixing orthographic errors.
- **Rloc-:** Redundancy is difficult to capture using just rule-based approaches and classifiers; our approach obtains 17.47 recall which places second among the 12 teams. The top system obtains 20.16 recall using a combination MT, LM, and rule-based method.
- **Wci:** Although there are 340 collocation errors, all teams performed poorly on this category. Our recall places 3rd behind two teams (AMU and CAMB) whose methods both used an MT system. Again, this demonstrates the difficulty of capturing whether a sentence is idiomatic through only classifiers and rule-based methods.

We note that our system obtains significantly higher precision than any of the top 10 teams in the 2014 Challenge (49.24 vs. 41.78), which comes at the expense of recall.

Limitations A key limitation of our method as well as most other translation-based methods is that it is trained on just parallel sentences, despite some errors requiring information about the surrounding text to make the proper correction. Even within individual sentences, when longer context is needed to make a correction (for example

⁶The team that placed 9th overall did not disclose their method; thus we only compare to the 12 remaining teams.

in many subject-verb agreement errors), the performance is hit-and-miss. The edits introduced by the system tend to be fairly local.

Other errors illustrate the need for natural language understanding, for example in Table 3.5 the correction *Broke my heart* \rightarrow *I broke my heart* and *I want to big size bag* \rightarrow *I want to be a big size bag*. Finally, although end-to-end approaches have the potential to fix a wide variety of errors, it is not straightforward to then classify the types of errors being made. Thus the system cannot easily provide error-specific feedback.

3.4.1 Aside: Extension to Speech

A natural question when considering the problem definition given in this chapter is whether the idea of correcting mistakes at the character-level can also be applied to speech, for example to correct phonetic errors that do not make sense in context or to add punctuation [8].

Until recently, speech systems operated by having an acoustic model along with a Hidden Markov Model. However, recent developments where a neural network is used to map directly from filter bank features to transcriptions allow us to use a neural character-level language model for the purpose of denoising speech transcriptions. Predictions over characters from the speech inputs are initially produced using a deep bidirectional recurrent neural network trained using the connectionist temporal classification (CTC) objective [43]. In this framework, the deep bidirectional recurrent neural network (DBRNN) models the probability over the character vocabulary at defined time windows over the speech input, as shown in Figure 3.3.

In order to handle various possible alignments between source and target, the CTC objective allows for blank tokens which are ignored and collapses repeat tokens that are not separated by blank tokens. Due to limited training data and the long input and output sequences involved in training such models, decoded outputs can often contain errors.

To remedy this, we propose introducing a neural character language model during decoding. The character language model places a prior over the output text, and to

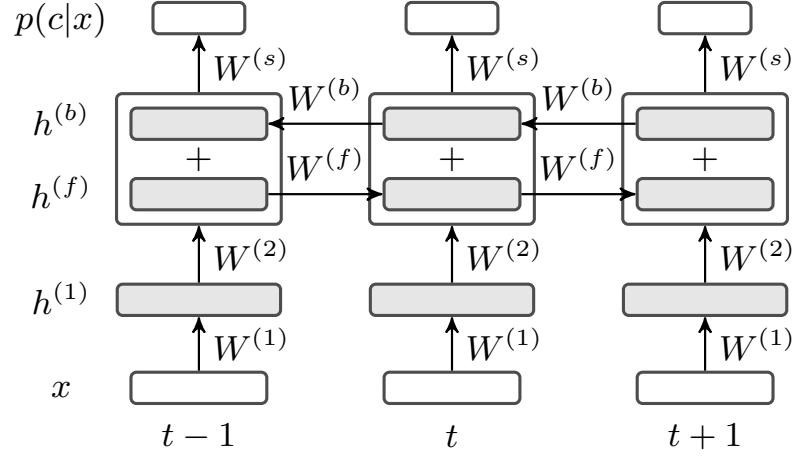


Figure 3.3: Deep bi-directional recurrent neural network to map input audio features X to a distribution over output characters at each timestep t . The network contains two hidden layers with the second layer having bi-directional temporal recurrence.

decode while taking language model probabilities into account, we use beam search to combine a character language model and the outputs of our DBRNN. Note that this search-based decoding method does not make a greedy approximation and instead assigns probability to a final hypothesis by integrating over all character sequences consistent with the hypothesis under our collapsing function $\kappa(\cdot)$. Algorithm [III](#) outlines the decoding procedure.

3.5 Related Work

Our work primarily builds on prior work on training encoder-decoder RNNs for machine translation [[59](#), [127](#), [22](#)]. The attention mechanism, which allows the decoder network to copy parts of the source sentence and cope with long inputs, is based on the content-based attention mechanism introduced by Bahdanau et al. [[6](#)], and the overall network architecture is based on that described by Chan et al. [[19](#)]. Our model is also inspired by character-level models as proposed by Graves [[44](#)]. More recent work has applied character-level models to machine translation and speech recognition as well, suggesting that it may be applicable to many other tasks that involve the problem of OOVs [[83](#), [91](#), [19](#)].

Algorithm 1 Beam Search Decoding: Given the likelihoods from our DBRNN and our character language model, for each time step t and for each string s in our current previous hypothesis set Z_{t-1} , we consider extending s with a new character. Blanks and repeat characters with no separating blank are handled separately. For all other character extensions, we apply our character language model when computing the probability of s . We initialize Z_0 with the empty string \emptyset . Notation: ζ' : character set excluding “_”, $s + c$: concatenation of character c to string s , $|s|$: length of s , $p_b(c|x_{1:t})$ and $p_{nb}(c|x_{1:t})$: probability of s ending and not ending in blank conditioned on input up to time t , $p_{\text{tot}}(c|x_{1:t})$: $p_b(c|x_{1:t}) + p_{nb}(c|x_{1:t})$

Inputs CTC likelihoods $p_{\text{ctc}}(c|x_t)$, character language model $p_{\text{clm}}(c|s)$

Parameters language model weight α , insertion bonus β , beam width k

Initialize $Z_0 \leftarrow \{\emptyset\}$, $p_b(\emptyset|x_{1:0}) \leftarrow 1$, $p_{nb}(\emptyset|x_{1:0}) \leftarrow 0$

for $t = 1, \dots, T$ **do**

$Z_t \leftarrow \{\}$

for s **in** Z_{t-1} **do**

$p_b(s|x_{1:t}) \leftarrow p_{\text{ctc}}(_|x_t)p_{\text{tot}}(s|x_{1:t-1})$

\triangleright Handle blanks

$p_{nb}(s|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{nb}(s|x_{1:t-1})$

\triangleright Handle repeat character collapsing

 Add s to Z_t

for c **in** ζ' **do**

$s^+ \leftarrow s + c$

if $c \neq s_{t-1}$ **then**

$p_{nb}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_{\text{tot}}(c|x_{1:t-1})$

else

$p_{nb}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_b(c|x_{1:t-1})$

\triangleright Repeat characters

 have “_” between

end if

 Add s^+ to Z_t

end for

end for

$Z_t \leftarrow k$ most probable s by $p_{\text{tot}}(s|x_{1:t})|s|^\beta$ in Z_t

\triangleright Apply beam

end for

Return $\arg \max_{s \in Z_t} p_{\text{tot}}(s|x_{1:T})|s|^\beta$

Treating grammatical error correction as a statistical machine translation problem is an old idea; the method of mapping “bad” to “good” sentences was used by many of the teams in the CoNLL 2014 Challenge [36, 57] as well as by earlier work such as Brockett et al. [16]. The work of Felice et al. [36] achieved the best $F_{0.5}$ -score of 37.33 in that year’s challenge using a combination of rule-based, language-model

ranking, and statistical machine translation techniques. Many other teams used a language model for re-ranking hypotheses as well. Other teams participating in the CoNLL 2014 Challenge used techniques ranging from rule-based systems to type-specific classifiers, as well as combinations of the two [117, 76]. The rule-based systems often focus on only a subset of the error types. The previous state of the art was achieved by Susanto [126] using the system combination method proposed by Heafield and Lavie [50] to combine three weaker systems.

Finally, our work uses data collected and shared through the generous efforts of the teams behind the CoNLL and Lang-8 datasets [98, 100, 102, 103]. Prior work has also proposed data augmentation for the grammar correction task [33, 116].

3.6 Conclusion

We present a neural network-based model for performing grammatical error correction. Our system is able correct errors on noisy data collected from an English learner forum and beats previous non-neural approaches performance on the CoNLL 2014 Challenge dataset of annotated essays. Key to our approach is the use of a character-based model with an attention mechanism, which allows for orthographic errors to be captured and avoids the OOV problem suffered by word-based neural machine translation methods.

Since this work, model architectures have improved, and other methods have been developed to avoid the OOV problem. But perhaps the most important finding from this chapter is that the data augmentation scheme we proposed did not generalize to other error distributions. If we could develop a better data synthesis technique, this would have great potential to improve the performance of such systems. Motivated by this finding, in the next chapter we try and better understand noise in the sequence modeling context.

Chapter 4

Noising Natural Language

Because there is so much noise in the world, people adopt rules of thumb. They share their rules of thumb with each other, and very few people have enough experience with interpreting noisy evidence to see that the rules are too simple.

Fischer Black, “Noise”

While in the previous chapter we considered denoising text, we now turn to better understanding the effects of noise in sequence modeling, in particular in natural language modeling. This is motivated by our observation that naively synthesizing article/determiner or singular vs. plural errors did not generalize to different error distributions. While our focus is on language modeling for AI writing assistants, language models are a crucial component in many domains, such as autocompletion, machine translation, and speech recognition, and in this chapter we find that noising can help for other tasks outside of error correction as well.

Recall that a key challenge when performing estimation in language modeling is the *data sparsity* problem: due to large vocabulary sizes and the exponential number of possible contexts, the majority of possible sequences are rarely or never observed, even for very short subsequences. In other application domains, data augmentation has been key to improving the performance of neural network models in the face of

insufficient data. In computer vision, for example, there exist well-established primitives for synthesizing additional image data, such as by rescaling or applying affine distortions to images [73, 66]. Similarly, in speech recognition adding a background audio track or applying small shifts along the time dimension has been shown to yield significant gains, especially in noisy settings [32, 47]. However, widely-adopted noising primitives have not yet been developed for neural network language models, despite some early work on noisy channel models for error correction [109, 115, 46] (see Chapter 3 for more on this).

Classic n -gram models of language cope with rare and unseen sequences by using smoothing methods, such as interpolation or absolute discounting [20]. Neural network models, however, have no notion of discrete counts, and instead use distributed representations to combat the curse of dimensionality [11]. Despite the effectiveness of distributed representations, overfitting due to data sparsity remains an issue. Existing regularization methods, however, are typically applied to weights or hidden units within the network [125, 72] instead of directly considering the input data.

In this chapter, we consider noising primitives and their effects on recurrent neural network-based language models. By examining the expected pseudocounts from applying the noising schemes, we draw connections between noising and linear interpolation smoothing. Using this connection, we then derive noising schemes that are analogues of more advanced smoothing methods. We demonstrate the effectiveness of these schemes for regularization through experiments on language modeling and machine translation. Finally, we validate our theoretical claims by examining the empirical effects of noising.

4.1 Method

4.1.1 Preliminaries

Let us briefly summarize the necessary background from Chapter 2 and also provide some more detail specific to this chapter. We consider language models where given

a sequence of indices $X = (x_1, x_2, \dots, x_T)$, over the vocabulary V , we model

$$p(X) = \prod_{t=1}^T p(x_t | x_{<t})$$

In n -gram models, it is not feasible to model the full context $x_{<t}$ for large t due to the exponential number of possible histories. Recurrent neural network (RNN) language models can (in theory) model longer dependencies, since they operate over distributed hidden states instead of modeling an exponential number of discrete counts [11, 96].

An L -layer recurrent neural network is modeled as $h_t^{(l)} = f_\theta(h_{t-1}^{(l)}, h_t^{(l-1)})$, where l denotes the layer index, $h^{(0)}$ contains the one-hot encoding of X , and in its simplest form f_θ applies an affine transformation followed by a nonlinearity. In this chapter, we use RNNs with a more complex form of f_θ , namely long short-term memory (LSTM) units [51], which have been shown to ease training and allow RNNs to capture longer dependencies. The output distribution over the vocabulary V at time t is $p_\theta(x_t | x_{<t}) = \text{softmax}(g_\theta(h_t^{(L)}))$, where $g : \mathbb{R}^{|h|} \rightarrow \mathbb{R}^{|V|}$ applies an affine transformation. The RNN is then trained by minimizing over its parameters θ the sequence cross-entropy loss $\ell(\theta) = -\sum_t \log p_\theta(x_t | x_{<t})$, thus maximizing the likelihood $p_\theta(X)$.

As an extension, we also consider encoder-decoder or sequence-to-sequence [22, 128] models where given an input sequence X and output sequence Y of length T_Y , we model

$$p(Y|X) = \prod_{t=1}^{T_Y} p(y_t | X, y_{<t}).$$

and minimize the loss $\ell(\theta) = -\sum_t \log p_\theta(y_t | X, y_{<t})$. This setting can also be seen as conditional language modeling, and encompasses tasks such as machine translation, where X is a source language sequence and Y a target language sequence, as well as language modeling, where Y is the given sequence and X is the empty sequence.

4.1.2 Smoothing and Noising

Recall that for a given context length l , an n -gram model of order $l + 1$ is optimal under the log-likelihood criterion. Hence in the case where an RNN with finite context achieves near the lowest possible cross-entropy loss, it behaves like an n -gram model.

Like n -gram models, RNNs are trained using maximum likelihood, and can easily overfit [144]. While generic regularization methods such L_2 -regularization and dropout are effective, they do not take advantage of specific properties of sequence modeling. In order to understand sequence-specific regularization, it is helpful to examine n -gram language models, whose properties are well-understood.

Smoothing for n -gram models When modeling $p(x_t|x_{<t})$, the maximum likelihood estimate $c(x_{<t}, x_t)/c(x_{<t})$ based on empirical counts puts zero probability on unseen sequences, and thus smoothing is crucial for obtaining good estimates. In particular, we consider interpolation, which performs a weighted average between higher and lower order models. The idea is that when there are not enough observations of the full sequence, observations of subsequences can help us obtain better estimates.¹ For example, in a bigram model, $p_{\text{interp}}(x_t|x_{t-1}) = \lambda p(x_t|x_{t-1}) + (1 - \lambda)p(x_t)$, where $0 \leq \lambda \leq 1$.

Noising for RNN models We would like to apply well-understood smoothing methods such as interpolation to RNNs, which are also trained using maximum likelihood. Unfortunately, RNN models have no notion of counts, and we cannot directly apply one of the usual smoothing methods. In this section, we consider two simple noising schemes which we proceed to show correspond to smoothing methods. Since we can noise the data while training an RNN, we can then incorporate well-understood generative assumptions that are known to be helpful in the domain. First consider the following two noising schemes:

- **unigram noising** For each x_i in $x_{<t}$, with probability γ replace x_i with a sample from the unigram frequency distribution.

¹For a thorough review of smoothing methods, see [20].

- **blank noising** For each x_i in $x_{<t}$, with probability γ replace x_i with a placeholder token “_”.

While blank noising can be seen as a way to avoid overfitting on specific contexts, we will see that both schemes are related to smoothing, and that unigram noising provides a path to analogues of more advanced smoothing methods.

4.1.3 Noising as Smoothing

We now consider the maximum likelihood estimate of n -gram probabilities estimated using the pseudocounts of the noised data. By examining these estimates, we draw a connection between linear interpolation smoothing and noising.

Unigram noising as interpolation To start, we consider the simplest case of bigram probabilities. Let $c(x)$ denote the count of a token x in the original data, and let $c_\gamma(x) \stackrel{\text{def}}{=} \mathbb{E}_{\tilde{x}}[c(\tilde{x})]$ be the expected count of x under the unigram noising scheme. We then have

$$\begin{aligned} p_\gamma(x_t|x_{t-1}) &= \frac{c_\gamma(x_{t-1}, x_t)}{c_\gamma(x_{t-1})} \\ &= [(1 - \gamma)c(x_{t-1}, x_t) + \gamma p(x_{t-1})c(x_t)]/c(x_{t-1}) \\ &= (1 - \gamma)p(x_t|x_{t-1}) + \gamma p(x_t), \end{aligned}$$

where $c_\gamma(x) = c(x)$ since our proposal distribution $q(x)$ is the unigram distribution, and the last line follows since $c(x_{t-1})/p(x_{t-1}) = c(x_t)/p(x_t)$ is equal to the total number of tokens in the training set. Thus we see that the noised data has pseudocounts corresponding to *interpolation* or a mixture of different order n -gram models with fixed weighting.

More generally, let $\tilde{x}_{<t}$ be noised tokens from \tilde{x} . We consider the expected prediction under noise

$$\begin{aligned} p_\gamma(x_t|x_{<t}) &= \mathbb{E}_{\tilde{x}_{<t}} [p(x_t|\tilde{x}_{<t})] \\ &= \sum_J \underbrace{\pi(|J|)}_{p(|J| \text{ swaps})} \sum_{x_K} \underbrace{p(x_t|x_J, x_K)}_{p(x_t|\text{noised context})} \prod_{z \in x_K} \underbrace{p(z)}_{p(\text{drawing } z)} \end{aligned}$$

where the mixture coefficients are $\pi(|J|) = (1 - \gamma)^{|J|} \gamma^{t-1-|J|}$ with $\sum_J \pi(|J|) = 1$. $J \subseteq \{1, 2, \dots, t-1\}$ denotes the set of indices whose corresponding tokens are left unchanged, and K the set of indices that were replaced.

Blank noising as interpolation Next we consider the blank noising scheme and show that it corresponds to interpolation as well. This also serves as an alternative explanation for the gains that other related work have found with the “word-dropout” idea [69, 30, 14]. As before, we do not noise the token being predicted x_t . Let $\tilde{x}_{<t}$ denote the random variable where each of its tokens is replaced by “_” with probability γ , and let x_J denote the sequence with indices J unchanged, and the rest replaced by “_”. To make a prediction, we use the expected probability over different noisings of the context

$$p_\gamma(x_t|x_{<t}) = \mathbb{E}_{\tilde{x}_{<t}} [p(x_t|\tilde{x}_{<t})] = \sum_J \underbrace{\pi(|J|)}_{p(|J| \text{ swaps})} \underbrace{p(x_t|x_J)}_{p(x_t|\text{noised context})},$$

where $J \subseteq \{1, 2, \dots, t-1\}$, which is also a mixture of the unnoised probabilities over subsequences of the current context. For example, in the case of trigrams, we have

$$p_\gamma(x_3|x_1, x_2) = \pi(2) p(x_3|x_1, x_2) + \pi(1) p(x_3|x_1, _) + \pi(1) p(x_3|_, x_2) + \pi(0) p(x_3|_, _)$$

where the mixture coefficient $\pi(i) = (1 - \gamma)^i \gamma^{2-i}$.

4.1.4 Borrowing Techniques

With the connection between noising and smoothing in place, we now consider how we can improve the two components of the noising scheme by considering:

1. Adaptively computing noising probability γ to reflect our confidence about a particular input subsequence.
2. Selecting a proposal distribution $q(x)$ that is less naive than the unigram distribution by leveraging higher order n -gram statistics.

Noising Probability Although it simplifies analysis, there is no reason why we should choose fixed γ ; we now consider defining an adaptive $\gamma(x_{1:t})$ which depends on the input sequence. Consider the following bigrams:

“and the”

“Humpty Dumpty”

The first bigram is one of the most common in English corpora; its probability is hence well estimated and should not be interpolated with lower order distributions. In expectation, however, using fixed γ_0 when noising results in the same lower order interpolation weight π_{γ_0} for common as well as rare bigrams. Intuitively, we should define $\gamma(x_{1:t})$ such that commonly seen bigrams are less likely to be noised.

The second bigram, “Humpty Dumpty,” is relatively uncommon, as are its constituent unigrams. However, it forms what [17] term a “sticky pair”: the unigram “Dumpty” almost always follows the unigram “Humpty”, and similarly, “Humpty” almost always precedes “Dumpty”. For pairs with high mutual information, we wish to avoid backing off from the bigram to the unigram distribution.

Let $N_{1+}(x_1, \bullet) \stackrel{\text{def}}{=} |\{x_2 : c(x_1, x_2) > 0\}|$ be the number of distinct continuations following x_1 , or equivalently the number of bigram types beginning with x_1 [20]. From the above intuitions, we arrive at the *absolute discounting* noising probability

$$\gamma_{\text{AD}}(x_1) = \gamma_0 \frac{N_{1+}(x_1, \bullet)}{\sum_{x_2} c(x_1, x_2)}$$

where for $0 \leq \gamma_0 \leq 1$ we have $0 \leq \gamma_{\text{AD}} \leq 1$, though in practice we can also clip larger noising probabilities to 1. Note that this encourages noising of unigrams that precede many possible other tokens while discouraging noising of common unigrams, since if we ignore the final token, $\sum_{x_2} c(x_1, x_2) = c(x_1)$.

Noised	$\gamma(x_{1:t})$	$q(x)$	Analogue
x_1	γ_0	$q(\text{"_"}) = 1$	interpolation
x_1	γ_0	unigram	interpolation
x_1	$\gamma_0 N_{1+}(x_1, \bullet) / c(x_1)$	unigram	absolute discounting
x_1, x_2	$\gamma_0 N_{1+}(x_1, \bullet) / c(x_1)$	$q(x) \propto N_{1+}(\bullet, x)$	Kneser-Ney

Table 4.1: **Noising schemes** Example noising schemes and their bigram smoothing analogues. Here we consider the bigram probability $p(x_1, x_2) = p(x_2|x_1)p(x_1)$. Notation: $\gamma(x_{1:t})$ denotes the noising probability for a given input sequence $x_{1:t}$, $q(x)$ denotes the proposal distribution, and $N_{1+}(x, \bullet)$ denotes the number of distinct bigrams in the training set where x is the first unigram. In all but the last case we only noise the context x_1 and not the target prediction x_2 .

Proposal Distribution While choosing the unigram distribution as the proposal distribution $q(x)$ preserves unigram frequencies, by borrowing from the smoothing literature we find another distribution performs better. We again begin with two motivating examples:

“San Francisco”

“New York”

Both bigrams appear frequently in text corpora. As a direct consequence, the unigrams “Francisco” and “York” also appear frequently. However, since “Francisco” and “York” typically follow “San” and “New”, respectively, they should not have high probability in the proposal distribution as they might if we use unigram frequencies [20]. Instead, it would be better to increase the proposal probability of unigrams with diverse histories, or more precisely unigrams that complete a large number of bigram types. Thus instead of drawing from the unigram distribution, we consider drawing from

$$q(x) \propto N_{1+}(\bullet, x)$$

Note that we now noise the prediction x_t in addition to the context $x_{1:t-1}$. Combining this new proposal distribution with the discounted $\gamma_{\text{AD}}(x_1)$ from the previous section, we obtain the noising analogue of Kneser-Ney smoothing.

Table 4.1 summarizes the discussed noising schemes. We provide pseudocode for the noising equivalent of Kneser-Ney smoothing in Algorithm 3.

4.1.5 Training and Testing

During training, noising is performed per batch and is done online such that each epoch of training sees a different noised version of the training data. At test time, to match the training objective we should sample multiple corrupted versions of the test data, then average the predictions [125]. In practice, however, we find that simply using the maximum likelihood (uncorrupted) input sequence works well; evaluation runtime remains unchanged.

4.1.6 Extensions

The schemes described are for the language model setting. To extend them to the sequence-to-sequence or encoder-decoder setting, we noise both $x_{<t}$ as well as $y_{<t}$. While in the decoder we have $y_{<t}$ and y_t as analogues to language model context and target prediction, it is unclear whether noising $x_{<t}$ should be beneficial. Empirically, however, we find this to be the case (Table 4.4).

4.2 Experiments

4.2.1 Language Modeling

Penn Treebank We train networks for word-level language modeling on the Penn Treebank dataset, using the standard preprocessed splits with a 10K size vocabulary [96]. The PTB dataset contains 929k training tokens, 73k validation tokens, and 82k test tokens. Following [144], we use minibatches of size 20 and unroll for 35 time steps when performing backpropagation through time. All models have two hidden layers and use LSTM units. Weights are initialized uniformly in the range $[-0.1, 0.1]$. We consider models with hidden sizes of 512 and 1500.

We train using stochastic gradient descent with an initial learning rate of 1.0, clipping the gradient if its norm exceeds 5.0. When the validation cross entropy does not decrease after a training epoch, we halve the learning rate. We anneal the learning rate 8 times before stopping training, and pick the model with the lowest perplexity

Algorithm 3 We provide pseudocode of the noising algorithm corresponding to bigram Kneser-Ney smoothing for n -grams (In the case of sequence-to-sequence tasks, we estimate the count-based parameters separately for source and target). To simplify, we assume a batch size of one. The noising algorithm is applied to each data batch during training. No noising is applied at test time.

Require counts $c(x)$, number of distinct continuations $N_{1+}(x, \bullet)$, proposal distribution $q(x) \propto N_{1+}(\bullet, x)$

Inputs X, Y batch of unnoised data indices, scaling factor γ_0

```

procedure NOISEBGKN( $X, Y$ )                                 $\triangleright X = (x_1, \dots, x_t), Y = (x_2, \dots, x_{t+1})$ 
   $\tilde{X}, \tilde{Y} \leftarrow X, Y$ 
  for  $j = 1, \dots, t$  do
     $\gamma \leftarrow \gamma_0 N_{1+}(x_j, \bullet) / c(x_j)$ 
    if  $\sim \text{Bernoulli}(\gamma)$  then
       $\tilde{x}_j \sim \text{Categorical}(q)$                                  $\triangleright$  Updates  $\tilde{X}$ 
       $\tilde{y}_j \sim \text{Categorical}(q)$ 
    end if
  end for
  return  $\tilde{X}, \tilde{Y}$                                             $\triangleright$  Run training iteration with noised batch
end procedure

```

Noising scheme	Validation	Test
Medium models (512 hidden size)		
none (dropout only)	84.3	80.4
blank	82.7	78.8
unigram	83.1	80.1
bigram Kneser-Ney	79.9	76.9
Large models (1500 hidden size)		
none (dropout only)	81.6	77.5
blank	79.4	75.5
unigram	79.4	76.1
bigram Kneser-Ney	76.2	73.4
[144]	82.2	78.4
[37] variational dropout (tied weights)	77.3	75.0
[37] (untied weights, Monte Carlo)	—	73.4

Table 4.2: Single-model perplexity on Penn Treebank with different noising schemes. We also compare to the variational method of [37], who also train LSTM models with the same hidden dimension. Note that performing Monte Carlo dropout at test time is significantly more expensive than our approach, where test time is unchanged.

on the validation set.

For regularization, we apply feed-forward dropout [111] in combination with our noising schemes. We report results in Table 4.2 for the best setting of the dropout rate (which we find to match the settings reported in [144]) as well as the best setting of noising probability γ_0 on the validation set.² Figure 4.1 shows the training and validation perplexity curves for a noised versus an unnoised run.

Our large models match the state-of-the-art regularization method for single model performance on this task. In particular, we find that picking $\gamma_{AD}(x_1)$ and $q(x)$ corresponding to Kneser-Ney smoothing yields significant gains in validation perplexity, both for the medium and large size models. Recent work [94, 145] has also achieved impressive results on this task by proposing different architectures which are orthogonal to our data augmentation schemes.

²Code will be made available at: <http://deeplearning.stanford.edu/noising>

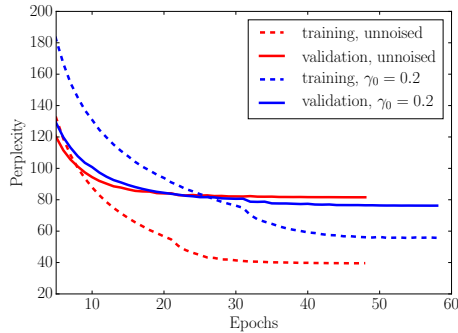
Noising scheme	Validation	Test
none	94.3	123.6
blank	85.0	110.7
unigram	85.2	111.3
bigram Kneser-Ney	84.5	110.6

Table 4.3: Perplexity on Text8 with different noising schemes.

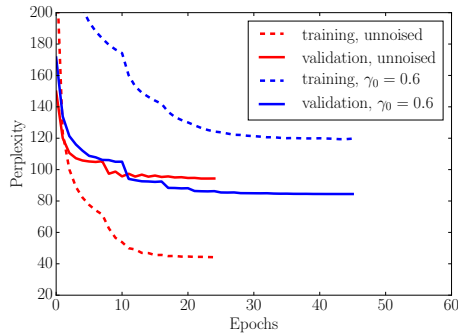
Scheme	Perplexity	BLEU
dropout, no noising	8.84	24.6
blank noising	8.28	25.3 (+0.7)
unigram noising	8.15	25.5 (+0.9)
bigram Kneser-Ney	7.92	26.0 (+1.4)
source only	8.74	24.8 (+0.2)
target only	8.14	25.6 (+1.0)

Table 4.4: Perplexities and BLEU scores for machine translation task. Results for bigram KN noising on only the source sequence and only the target sequence are given as well.

Text8 In order to determine whether noising remains effective with a larger dataset, we perform experiments on the significantly larger Text8 corpus^B. The first 90M characters are used for training, the next 5M for validation, and the final 5M for testing, resulting in 15.3M training tokens, 848K validation tokens, and 855K test tokens. We preprocess the data by mapping all words which appear 10 or fewer times to the unknown token, resulting in a 42K size vocabulary. Other parameter settings are the same as described in the Penn Treebank experiments, besides that only models with hidden size 512 are considered, and noising is not combined with feed-forward dropout. Results are given in Table 4.3.



(a) Penn Treebank corpus.



(b) Text8 corpus.

Figure 4.1: Example training and validation curves for an unnoised model and model regularized using the bigram Kneser-Ney noising scheme.

4.2.2 Machine Translation

For our machine translation experiments we consider the English-German machine translation track of IWSLT 2015³. The IWSLT 2015 corpus consists of sentence-aligned subtitles of TED and TEDx talks. The training set contains roughly 190K sentence pairs with 5.4M tokens. Following [86], we use TED tst2012 as a validation set and report BLEU score results [108] on tst2014. We limit the vocabulary to the top 50K most frequent words for each language.

We train a two-layer LSTM encoder-decoder network [128, 22] with 512 hidden units in each layer. The decoder uses an attention mechanism [6] with the dot alignment function [87]. The initial learning rate is 1.0 and we start halving the learning

³<http://mattmahoney.net/dc/text8.zip>

⁴<http://workshop2015.iwslt.org/>

rate when the relative difference in perplexity on the validation set between two consecutive epochs is less than 1%. We follow training protocols as described in [128]: (a) LSTM parameters and word embeddings are initialized from a uniform distribution between $[-0.1, 0.1]$, (b) inputs are reversed, (c) batch size is set to 128, (d) gradient clipping is performed when the norm exceeds a threshold of 5. We set hidden unit dropout rate to 0.2 across all settings as suggested in [87]. We compare unigram, blank, and bigram Kneser-Ney noising. Noising rate γ is selected on the validation set.

Results are shown in Table 4.4. We observe performance gains for both blank noising and unigram noising, giving roughly +0.7 BLEU score on the test set. The proposed bigram Kneser-Ney noising scheme gives an additional performance boost of +0.5-0.7 on top of the blank noising and unigram noising models, yielding a total gain of +1.4 BLEU.

4.3 Discussion

4.3.1 Scaling γ via Discounting

We now examine whether discounting has the desired effect of noising subsequences according to their uncertainty. If we consider the discounting

$$\gamma_{AD}(x_1) = \gamma_0 \frac{N_{1+}(x_1, \bullet)}{c(x_1)}$$

we observe that the denominator $c(x_1)$ can dominate than the numerator $N_{1+}(x_1, \bullet)$. Common tokens are often noised infrequently when discounting is used to rescale the noising probability, while rare tokens are noised comparatively much more frequently, where in the extreme case when a token appears exactly once, we have $\gamma_{AD} = \gamma_0$. Due to word frequencies following a Zipfian power law distribution, however, common tokens constitute the majority of most texts, and thus discounting leads to significantly less noising.

We compare the performance of models trained with a fixed γ_0 versus a γ_0 rescaled using discounting. As shown in Figure 4.2, bigram discounting leads to gains in

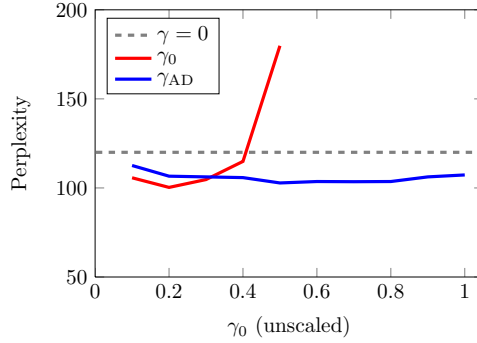


Figure 4.2: Perplexity with noising on Penn Treebank while varying the value of γ_0 . Using discounting to scale γ_0 (yielding γ_{AD}) maintains gains for a range of values of noising probability, which is not true for the unscaled case.

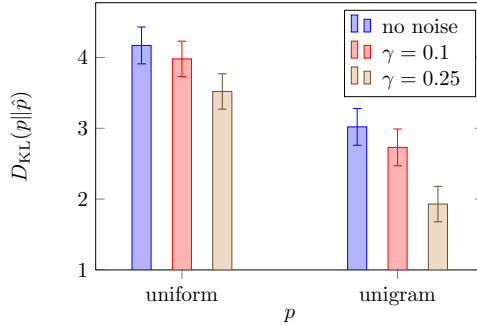


Figure 4.3: Mean KL-divergence over validation set between softmax distributions of noised and unnoised models and lower order distributions. Noised model distributions are closer to the uniform and unigram frequency distributions.

perplexity for a much broader range of γ_0 . Thus the discounting ratio seems to effectively capture the “right” tokens to noise.

4.3.2 Noised versus Unnoised Models

Smoothed distributions In order to validate that data noising for RNN models has a similar effect to that of smoothing counts in n -gram models, we consider three models trained with unigram noising as described in Section 4.2.1 on the Penn Treebank corpus with $\gamma = 0$ (no noising), $\gamma = 0.1$, and $\gamma = 0.25$. Using the trained

Noising	Bigrams	Trigrams
none (dropout only)	2881	381
blank noising	2760	372
unigram noising	2612	365

Table 4.5: Perplexity of last unigram for unseen bigrams and trigrams in Penn Treebank validation set. We compare noised and unnoised models with noising probabilities chosen such that models have near-identical perplexity on full validation set.

models, we measure the Kullback-Leibler divergence $D_{\text{KL}}(p||q) = \sum_i p_i \log(p_i/q_i)$ over the validation set between the predicted softmax distributions, \hat{p} , and the uniform distribution as well as the unigram frequency distribution. We then take the mean KL divergence over all tokens in the validation set.

Recall that in interpolation smoothing, a weighted combination of higher and lower order n -gram models is used. As seen in Figure 4.3, the softmax distributions of noised models are significantly closer to the lower order frequency distributions than unnoised models, in particular in the case of the unigram distribution, thus validating our analysis in Section 4.1.3.

Unseen n -grams Smoothing is most beneficial for increasing the probability of unobserved sequences. To measure whether noising has a similar effect, we consider bigrams and trigrams in the validation set that do not appear in the training set. For these unseen bigrams (15062 occurrences) and trigrams (43051 occurrences), we measure the perplexity for noised and unnoised models with near-identical perplexity on the full set. As expected, noising yields lower perplexity for these unseen instances.

4.4 Related Work

Our work can be viewed as a form of data augmentation, for which to the best of our knowledge there exists no widely adopted schemes in language modeling with neural networks. Classical regularization methods such as L_2 -regularization are typically applied to the model parameters, while dropout is applied to activations which

can be along the forward as well as the recurrent directions [144, 119, 37]. Others have introduced methods for recurrent neural networks encouraging the hidden activations to remain stable in norm, or constraining the recurrent weight matrix to have eigenvalues close to one [67, 2, 72]. These methods, however, all consider weights and hidden units instead of the input data, and are motivated by the vanishing and exploding gradient problem.

Feature noising has been demonstrated to be effective for structured prediction tasks, and has been interpreted as an explicit regularizer [136]. Additionally, [134] show that noising can inject appropriate generative assumptions into discriminative models to reduce their generalization error, but do not consider sequence models [135].

The technique of randomly zero-masking input word embeddings for learning sentence representations has been proposed by [53], [69], and [30], and adopted by others such as [14]. However, to the best of our knowledge, no analysis has been provided besides reasoning that zeroing embeddings may result in a model ensembling effect similar to that in standard dropout. This analysis is applicable to classification tasks involving sum-of-embeddings or bag-of-words models, but does not capture sequence-level effects. [10] also make an empirical observation that the method of randomly replacing words with fixed probability with a draw from the uniform distribution improved performance slightly for an image captioning task; however, they do not examine why performance improved.

4.5 Conclusion

In this chapter, we show that data noising is effective for regularizing neural network-based sequence models. By deriving a correspondence between noising and smoothing, we are able to adapt advanced smoothing methods for n -gram models to the neural network setting, thereby incorporating well-understood generative assumptions of language. Possible applications include examining how these techniques generalize to sequence modeling in other domains, or exploring noising for improving performance in low resource settings—which is what we’ll look at in the next two chapters.

Chapter 5

Noising & Denoising Natural Language

After obtaining a better understanding of the effects of noise in neural sequence models, we return to the problem we addressed in Chapter 3 and use those insights to training better denoising models. Recall that classifier-based approaches to error correction are limited in their ability to capture a broad range of error types [103]. Machine translation-based approaches—that instead translate noisy, ungrammatical sentences to clean, corrected sentences—can flexibly handle a large variety of errors; however, such approaches are bottlenecked by the need for a large dataset of source-target sentence pairs.

To address this data sparsity problem (once more), we again explore methods for synthesizing noisy sentences from clean sentences, thus generating an additional artificial dataset of noisy and clean sentence pairs. A simple approach to noise clean text is to noise individual tokens or bigrams, for example by replacing each token with a random draw from the unigram distribution as described in Chapter 4. This type of approach, however, tends to generate highly unrealistic noise and fails to capture phrase-level phenomena. Other rule-based approaches fail to capture a diverse set of error types, as we found in Chapter 3.

Thus, we consider a method inspired by the backtranslation procedure for machine translation [121]. Our method combines a neural sequence transduction trained on

a seed corpus of clean→noisy pairs with *beam search noising* procedures to produce more diversity in the decoded outputs. This technique addresses two issues with existing synthesis techniques for grammar correction:

1. By using a neural model trained end-to-end on a large corpus of noisy and clean sentences, the model is able to generate rich, diverse errors that better capture the noise distribution of real data.
2. By encouraging diversity through applying noise to hypotheses during decoding, we avoid what we refer to as the *one-to-many* problem, where decoding from a model trained on clean→noisy examples results in overly clean output, since clean subphrases still form the majority of noisy examples.

We perform experiments using several noising methods to validate these two claims, yielding gains on two benchmarks. Our main empirical result is that, starting with only clean news data and models trained on a parallel corpus of roughly 1.3 million sentences, we can train models with additional synthesized data that nearly match the performance of models trained on 3 million nonsynthesized examples.

5.1 Method

We first briefly describe the neural model we use, then detail the noising schemes we apply when synthesizing examples.

5.1.1 Model

In order to generate noisy examples as well as to translate ungrammatical examples to their corrected counterparts, our method uses two neural encoder-decoder models:

1. The first is the *noising* model, which, given a clean sentence, is used to generate a noised version of that sentence. This model is trained on a *seed corpus* of parallel clean→noisy sentences.
2. The second is the *denoising* model, which, given a noisy, ungrammatical sentence, generates the clean, corrected sentence.

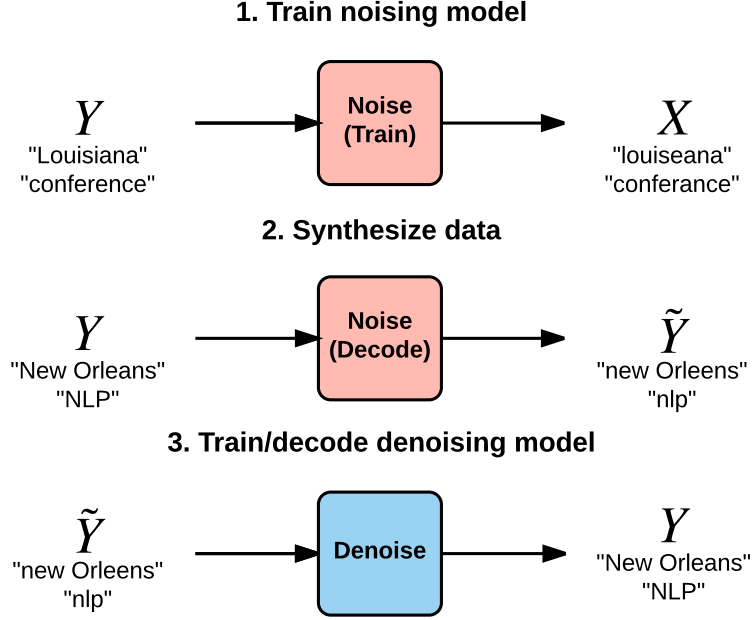


Figure 5.1: Overview of method. We first train a noise model on a seed corpus, then apply noise during decoding to synthesize data that is in turn used to train the denoising model.

For both models, we use the same convolutional encoder-decoder to model

$$p(Y|X) = \prod_{t=1}^{T_Y} p(y_t|X, y_{1:t-1}; \theta)$$

where $X = (x_1, x_2, \dots, x_{T_X})$ is the source sequence and $Y = (y_1, y_2, \dots, y_{T_Y})$ the corresponding target sequence, and we minimize the usual training loss

$$\ell(\theta) = -\log \sum_{t=1}^{T_Y} p(y_t|X, y_{1:t-1}; \theta)$$

thus maximizing log-likelihood. The model architecture we use is similar to that described by Kalchbrenner et al. [60] and Gehring et al. [40]. Gated convolutions are applied with masking—to avoid peeking at future inputs when training using teacher forcing—such that they form an autoregressive network similar to a recurrent neural network with gated hidden units. This architecture was selected so that training

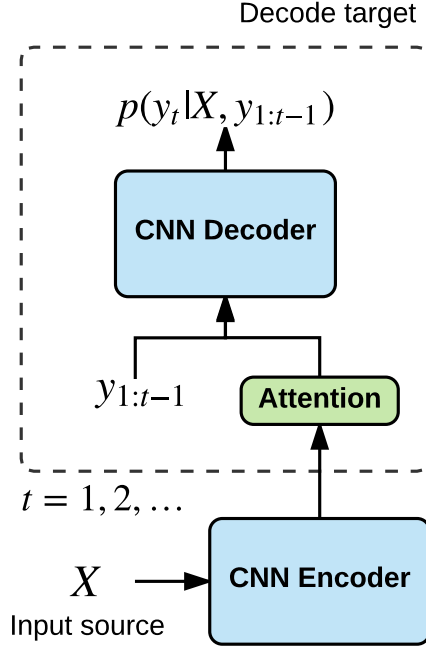


Figure 5.2: Model architecture used for both noising and denoising networks.

steps could be parallelized across the time dimension through the use of convolutions. However, we emphasize that the architecture is not a focus of this paper, and we would expect that RNN architectures with LSTM cells would achieve similar results, and more recent architectures may work even better [13]. For simplicity and to avoid handling out-of-vocabulary words, we use byte-pair encoding tokenization [14]. Figure 5.2 illustrates the model architecture.

5.1.2 Noising

The amount of parallel data is often the limiting factor in the performance of neural network systems. In order to obtain more parallel examples for the grammar correction task, we take clean text Y and apply noise, yielding noisy text \tilde{Y} , then train a denoising model to map from \tilde{Y} back to Y . The noising process used to generate \tilde{Y} greatly affects final performance. First, we consider noising methods which we use as our baselines, as well as the drawbacks for each method.

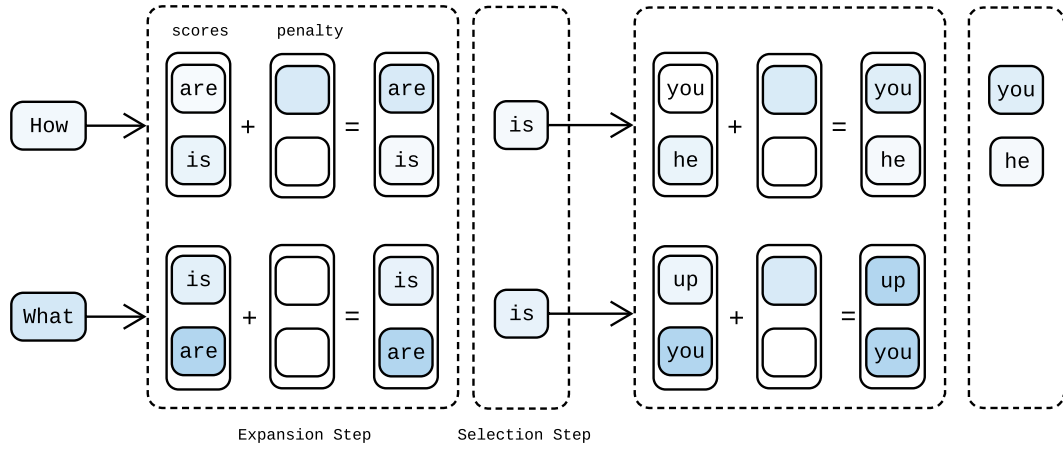


Figure 5.3: Illustration of random noising with beam width 2. Darker shading indicates less probable expansions. In this example, greedy decoding would yield “How are you”. Applying noise penalties, however, results in the hypotheses “How is you/he”. Note that applying a penalty does not always result in an expansion falling off the beam.

- **appending clean examples:** We first consider simply appending clean examples with no noise applied to both the source and the target. The aim is for the decoder to learn a better language model when trained on additional clean text, similar to the motivation described in Dai and Le [30]. However, for the models we consider, the attention mechanism allows copying of source to target. Thus the addition of examples where source and target are identical data may also cause the model to become too conservative with edits and thus reduce the recall of the system.
- **token noising:** Here we simply consider a context window of at most two characters or words and allow word/character deletions and transpositions.

First, for every *character* in each word we sample deletions, followed by transpositions. Then we sample deletions and transpositions for every *word* in the sentence. Deletion and transposition probabilities were selected such that overall character and word-level edit distances roughly matched the edit distances between clean and noisy examples in our parallel seed corpus. While this method

is fast to apply, it tends to produce highly unrealistic errors leading to a mismatch between the synthesized and real parallel data.

- **reverse noising:** For reverse noising, we simply train a reverse model from $Y \rightarrow X$ using our parallel noisy-clean corpus and run standard beam search to generate noisy targets \tilde{Y} from clean inputs Y . However, we find vanilla reverse noising tends to be too conservative. This is due to the *one-to-many* problem where a clean sentence has many possible noisy outputs which mostly consist of clean phrases. The output then contains far fewer errors on average than the original noisy text.

To address the drawback of the reverse noising scheme, we draw inspiration from ideas for increasing diversity of outputs in dialogue [78]. During the beam search procedure, we add noise to the scores of hypotheses on the beam to encourage decoding to stray from the greedy output. Recall that during beam search (Chapter 2), we iteratively grow a set of hypotheses $\mathcal{H} = \{h_1, h_2, \dots\}$, only keeping the top hypotheses after each step of decoding according to some scoring function $s(h)$. Extending the reverse noising scheme, the *beam search noising* schemes we consider are:

- **rank penalty noising** We directly apply the method of Li et al. [78]. At every step of the search procedure, siblings from the same parent are penalized by adding $k\beta_{\text{rank}}$ to their scores, where k is their rank (in descending log-likelihood) amongst their siblings and β_{rank} is a penalty hyperparameter corresponding to some log-probability.
- **top penalty noising** Only the top (most-probable) hypothesis h_{top} of the beam is penalized by adding β_{top} to its score $s(h_{\text{top}})$.
- **random noising** Every hypothesis is penalized by adding $r\beta_{\text{random}}$ to its score, where r is drawn uniformly from the interval $[0, 1]$. For sufficiently large β_{random} , this leads to a random shuffling of the ranks of the hypotheses according to their scores.

An illustration of the random noising algorithm is shown in Figure 5.3. Note that although rank penalty noising should encourage hypotheses whose parents have

similar scores to remain on the beam, it can also tend to leave the hypothesis from greedy decoding on the beam in the case where softmax output distributions are highly peaked. This is much more of an issue for tasks that involve significant copying of source to target, such as grammar correction. Note also that the random noising can yield more diverse outputs than top penalty noising, depending on the probability with which each is applied. All of the beam search noising methods described are intended to increase the diversity and the amount of noise in the synthesized outputs \tilde{Y} . By performing beam search noising, we can produce errors such as those shown in Table 5.4.

5.1.3 Denoising

Once noised data has been generated, *denoising* simply involves using a neural sequence transduction model to backtranslate the noised text to the original clean text. For denoising, during decoding we apply length normalization as well as a coverage penalty to the scoring function $s(h)$ [137] as detailed in Chapter 2. The final scoring function also incorporates a 5-gram language model trained on a subset of Common Crawl, estimated with Kneser-Ney smoothing using KenLM [49].

5.2 Experiments

To determine the effectiveness of the described noising schemes, we synthesize additional data using each and evaluate the performance of models trained using the additional data on two benchmarks.

Datasets For training our sequence transduction models, we combine the publicly available English Lang-8 dataset, a parallel corpus collected from a language learner forum, with training data from the CoNLL 2014 challenge [98, 103]. We refer to this as the “base” dataset. Junczys-Dowmunt and Grundkiewicz [58] additionally scraped 3.3M pairs of sentences from Lang-8. Although this expanded dataset, which we call the “expanded” dataset, is not typically used when comparing performance on grammar correction benchmarks, we use it instead to compare performance when

Corpus	Sent. Pairs
CoNLL 2014	60K
Lang-8	1.3M
Lang-8 expanded	3.3M
synthesized (NYT 2007)	1.0M
base (CoNLL + L8)	1.3M
expanded (CoNLL + L8 expanded)	3.3M

Table 5.1: Summary of training corpora.

Method	Dev (no LM)			Dev			Test		
	<i>P</i>	<i>R</i>	<i>F</i> _{0.5}	<i>P</i>	<i>R</i>	<i>F</i> _{0.5}	<i>P</i>	<i>R</i>	<i>F</i> _{0.5}
none	50.7	10.5	28.7	48.4	17.2	35.5	52.7	27.5	44.5
clean	56.1	9.4	28.1	47.5	16.9	34.8	52.3	27.5	44.3
token	49.7	11.9	30.4	47.7	18.7	36.4	51.4	30.3	45.1
reverse	53.1	13.0	32.8	50.5	19.1	38.0	54.7	29.6	46.8
rank	51.3	12.3	31.4	51.0	18.3	37.6	54.3	29.3	46.4
top	49.1	17.4	36.0	47.7	23.9	39.8	50.9	34.7	46.6
random	50.0	17.9	36.8	48.9	23.0	39.9	54.2	35.4	49.0
expanded	64.4	11.2	33.0	54.9	20.0	40.7	57.2	32.0	49.4
Yuan and Briscoe [143]	—	—	—	—	—	—	—	—	39.9
Ji et al. [54]	—	—	28.6	—	—	33.5	—	—	45.2
Junczys-Dowmunt et al. (2016)	—	—	—	—	—	—	61.3	28.0	49.5
Chollampatt and Ng [24]	—	—	—	—	—	—	65.5	33.1	54.8

Table 5.2: Results on CoNLL 2013 (Dev) and CoNLL 2014 (Test) sets. All results use the “base” parallel corpus of 1.3M sentence pairs along with additional synthesized data (totaling 2.3M sentence pairs) except for “expanded”, which uses 3.3M nonsynthesized sentence pairs (and no synthesized data).

training on additional synthesized data versus nonsynthesized data. For clean text to be noised, we use the LDC New York Times corpus for 2007, which yields roughly 1 million sentences. A summary of the data used for training is given in Table 5.1.

We use the CoNLL 2013 evaluation set as our development set in all cases [102]. Our test sets are the CoNLL 2014 evaluation set and the JFLEG test set [103, 101]. Because CoNLL 2013 only has a single set of gold annotations while CoNLL 2014 has two, performance metrics tend to be significantly higher on CoNLL 2014. As in Chapter 3, we report precision, recall, and $F_{0.5}$ score. On JFLEG, we report results

with the GLEU metric (similar to BLEU) developed for the dataset.

Training and decoding details All models are trained using stochastic gradient descent with annealing based on validation perplexity on a small held-out subset of the Lang-8 corpus. We apply both dropout and weight decay regularization. We observed that performance tended to saturate after 30 epochs. Decoding is done with a beam size of 8; in early experiments, we did not observe significant gains with larger beam sizes [65].

5.2.1 CoNLL

Results for the CoNLL 2013 (dev) and 2014 (test) datasets but with and without language model reranking are given in Table 5.2. In general, adding noised data helps, while simply adding clean data leads the model to be too conservative. Overall, we find that the random noising scheme yields the most significant gain of 4.5 F -score. Surprisingly, we find that augmenting the base dataset with synthesized data generated with random noising yields nearly the same performance when compared to using only nonsynthesized examples. To determine whether this might be due to overfitting, we reduced the dropout rate when training on the “expanded” dataset, but did not observe better results.

The random noising scheme achieves the best performance, while the top noising scheme matches the best performance on the development set but not the test set. We believe this is due to a mismatch between the CoNLL 2013 dev and 2014 test sets. Since the 2013 dev set has only a single annotator, methods are encouraged to target higher recall, such that the top noising scheme was optimized for precision over recall. To check this, we ran decoding on CoNLL 2014 using the best dev settings with no language model, and found that the top noising scheme yielded an $F_{0.5}$ -score of 45.2, behind only random (47.1) and ahead of token (42.0) and reverse (43.9) noising. Overall, we find the data synthesis method we describe to yield large gains in recall.

For completeness, we also compare to other state-of-the-art systems, such as the phrase-based machine translation system by Junczys-Dowmunt and Grundkiewicz

[58], who performed parameter tuning with sparse and dense features by cross-validation on the CoNLL 2014 training set. Chollampatt and Ng [24] achieve even higher state-of-the-art results using the neural machine translation model of Gehring et al. [40] along with improvements to the reranking procedure.

5.2.2 JFLEG

Recently, Napoles et al. [100] introduced the JFLEG dataset, intended to evaluate the fluency of grammar correction systems rather than simply the precision and recall of edits. The evaluation metric proposed is GLEU, a variant of BLEU score. Most results for this task were reported with hyperparameter settings from the CoNLL task; hence we report results with the best settings on our CoNLL 2013 dev set. Results are shown in Table 5.3¹. Token noising performs surprisingly well; we suspect this is because a significant portion of errors in the JFLEG dataset are spelling errors, as demonstrated from strong gains in performance by using a spelling checker reported by Chollampatt and Ng [24].

5.3 Discussion

Our experiments illustrate that synthesized parallel data can yield large gains on the grammar correction task. However, what factors make for an effective data synthesis technique? We consider the properties of the noising scheme and the corresponding data that lead to better performance.

5.3.1 Realism and Human Evaluation

First, we manually compare each of the different noising methods to evaluate how “realistic” the errors introduced are. This is reminiscent of the generative adversarial network setting [41], where the generator seeks to produce samples that fool the discriminator. Here the discriminator is a human evaluator who, given the clean sentence Y , tries to determine which of two sentences X and \tilde{Y} is the true noisy

¹Comparisons taken from <https://github.com/keisks/jfleg>

Scheme	P	R	$F_{0.5}$	GLEU
none	68.9	44.2	62.0	53.9
clean	69.2	42.8	61.6	54.1
token	69.2	47.6	63.5	55.9
reverse	69.1	42.1	61.3	53.8
rank	68.3	43.3	61.2	54.4
top	67.3	48.2	62.4	55.5
random	69.1	48.5	63.7	56.6
expanded	72.7	45.9	65.1	56.2
Sakaguchi et al. [118] [†]				54.0
Ji et al. [54]				53.4
Yuan and Briscoe [143]				52.1
Junczys-Dowmunt et al. (2016)				51.5
Chollampatt and Ng [24]				57.5

Table 5.3: Results on the JFLEG test set (we use best hyperparameter settings from CoNLL dev set). GLEU is a variant of BLEU developed for this task; higher is better [100]. [†]Tuned to JFLEG dev set.

sentence, and which is the synthesized sentence. To be clear, we do not train with a discriminator—the beam search noising procedures we proposed alone are intended to yield convincing errors.

For each noising scheme, we took 100 (X, Y) pairs from the development set (500 randomly chosen pairs combined), then generated \tilde{Y} from Y . We then shuffled the examples and the order of X and \tilde{Y} such that the identity of X and \tilde{Y} *as well as the noising scheme* used to generate \tilde{Y} were unknown². Given Y , the task for human evaluators is to predict whether X or \tilde{Y} was the synthesized example. For every example, we had two separate evaluators label the sentence they thought was synthesized. We chose to do this labeling task ourselves (blind to system) since we were familiar with the noising schemes used to generate examples, which should reduce the number of misclassifications. Results are shown in Figure 5.4, and examples of the evaluation task are provided in Table 5.4.

²Hence the human labelers cannot favor a particular scheme unless it can be distinguished from \tilde{Y} .

	Sentence	1 or 2
clean	Day after day , I get up at 8 o'clock .	
1	I got up at 8 o'clock day after day .	
2	Day after day , I get up 8 o'clock in the week .	
clean	Thanks Giving Day in Korea is coming soon .	
1	In Korea , it 's coming soon , thanks Giving day .	
2	Thanks Giving Day in korea is coming soon .	
clean	After I practiced , I could play the song perfectly .	
1	After the results , I could accomplish without a fault .	
2	When I tried that , I could play the song perfectly .	
clean	Currently , I 'm studying to take the TOEIC exam for my future career .	
1	I am studying to take TOEIC exam for career of my future .	
2	Currently , I will have take TOEIC exam for future career .	
clean	There is one child who is 15 years old and a mother who is around 50 .	
1	There are one child who is 15 years old and mother is around 50 .	
2	It has one child , 15 years old and the mother who is around 50 years old .	
clean	But at the beginning , I suffered from a horrible pain in my jaw .	
1	But at the first time , I suffer from a horrible pain on my jaw .	
2	But at the beginning , I suffered from a horrible pain in my jaw joint .	

Table 5.4: Examples of nonsynthesized and synthesized sentences from validation set. Which example (1 or 2) was synthesized? Answers: 1, 1, 2, 1, 2, 1

5.3.2 Noise Frequency and Diversity

Comparing the performance using different noising methods on the CoNLL 2014 dataset to the human evaluation in the previous section, we see that generating errors which *match* the real distribution tends to result in higher performance, as seen by the poor performance of token noising relative to the other methods. Injecting the appropriate *amount* of noise is important as well, as seen by improved performance when using beam search noising to increase diversity of outputs, and no performance gain when simply adding clean text.

We observe that token noising, despite matching the frequency of errors, fails to generate realistic errors (Figure 5.4). On the other hand, reverse noising yields significantly more convincing errors, but the edit distance between synthesized examples is significantly lower than in real data (Figure 5.5). A combination of sufficient amounts

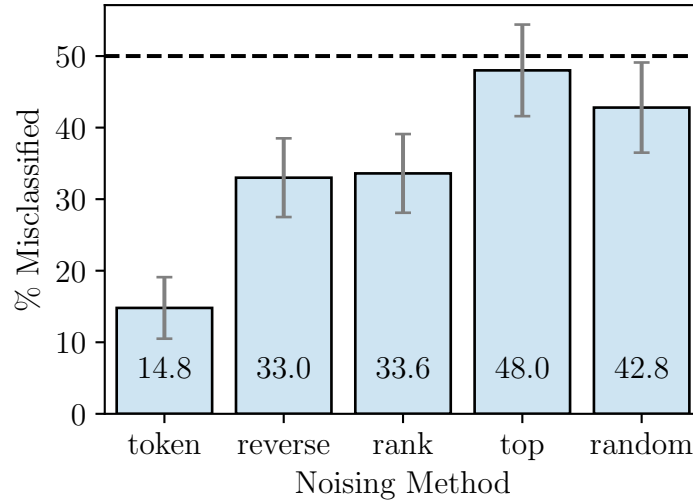


Figure 5.4: Percentage of time human evaluators misclassified synthesized noisy sentence \hat{Y} (vs. X) when using each noising scheme, along with 95% confidence intervals. The best we can expect any scheme to do is 50%.

of noise and rich, diverse errors appears to lead to better model performance.

5.3.3 Error Type Distribution Mismatch

Mismatches in the distribution of error types can often severely impact the performance of data synthesis techniques for grammar correction [34]. For example, only synthesizing noun number articles or preposition errors based on rules may improve the performance for those two error types, but may hurt overall performance, as we observed in Chapter 3. In contrast, the approaches we consider, with the exception of token noising, are fully data-driven, and hence we would expect gains across all different error types. We observe this is the case for random noising, as shown in Figure 5.6.

5.3.4 Data Sparsity and Domain Adaptation

Domain adaptation can yield significant differences in performance for dissimilar domains (such as those of the datasets used in our experiments) [31]. The Lang-8, CoNLL, and JFLEG datasets contain online forum data and essay data from English

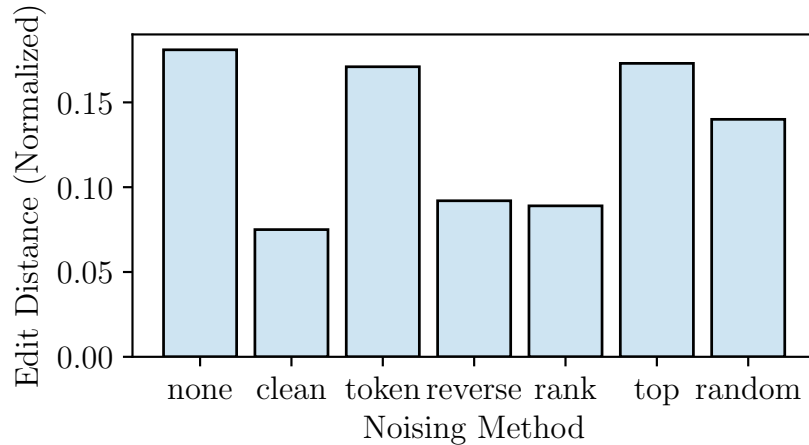


Figure 5.5: Mean edit distance between sentence pairs in X and Y *after* augmentation with noised sentences. *none* contains no synthesized examples while *clean* refers to the baseline of simply appending clean examples (source = target).

learners. The n -gram language model is estimated using Common Crawl data from the web. The clean data which we noise is collected from a news corpus. Yet each dataset yields significant gains. This suggests that at current levels of system performance, data sparsity remains the key data issue, more so than domain adaptation.

It is also possible that LDC New York Times data is better matched to the CoNLL essay data than the Lang-8 forum data, and this in part accounts for the large gains we observe from training on synthesized data.

5.4 Related Work

Noising While for images, there are natural noising primitives such as rotations, small translational shifts, and additive Gaussian noise, similar primitives are not as well developed for text data. Similarly, while denoising autoencoders for images have been shown to help with representation learning [132], similar methods for learning representations are not well developed for text. Some recent work has proposed noising—in the form of dropping or replacing individual tokens—as a regularizer when training sequence models, where it has been demonstrated to have a smoothing effect on the softmax output distribution [14, 140, 30, 69].

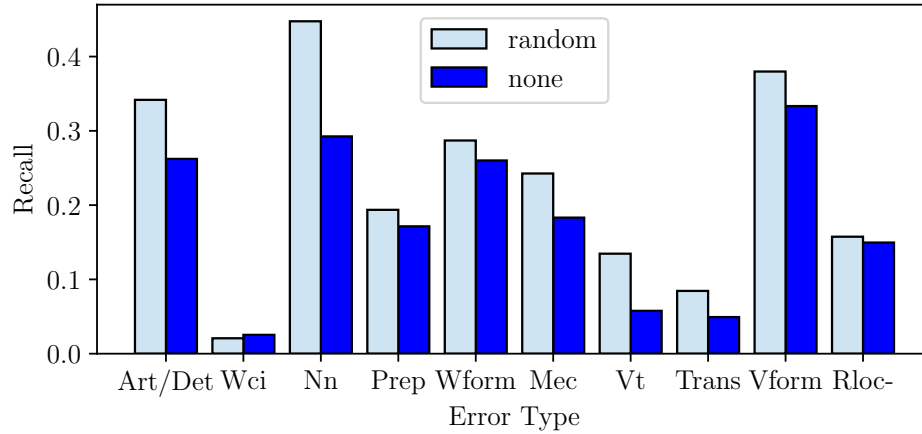


Figure 5.6: Recall vs. error type for the ten most frequent error types in our dev set. Noising improves recall uniformly across error types (See Ng et al. [113] for a description of error types).

Grammar correction Recent work by Chollampatt and Ng [24] has achieved impressive performance on the benchmarks we consider using convolutional encoder-decoder models. Previous work (such as that described in Chapter 3) using data synthesis for grammatical error correction (GEC) has introduced errors by examining the distribution of error types, then applying errors according to those distributions together with lexical or part-of-speech features based on a small context window [16, 34]. While these methods can introduce many possible edits, they are not as flexible as our approach inspired by the backtranslation procedure for machine translation [121]. This is important as neural language models not explicitly trained to track long-range linguistic dependencies can fail to capture even simple noun-verb errors [84]. Recently, in the work perhaps most similar to ours, Rei et al. [113] propose using statistical machine translation and backtranslation along with syntactic patterns for generating errors, albeit for the error detection task.

Neural machine translation Recent end-to-end neural network-based approaches to machine translation have demonstrated strong empirical results [128, 22]. Building off of these strong results on machine translation, we use neural encoder-decoder models with attention [6] for both our data synthesis (noising) and grammar correction (denoising) models. Although many recent works on NMT have focused on

improving the neural network architecture, the model architecture is orthogonal to the contributions in this work, where we instead focus on data synthesis. In parallel to our work, work on machine translation without parallel corpora has also explored applying noise to avoid copying when pretraining autoencoders by swapping adjacent words [70, 4].

Diverse decoding Key to the data generation procedure we describe is adding noise to the scores of hypotheses during beam search—otherwise, decoded outputs tend to contain too few errors. This is inspired by work in dialogue, in which neural network models tend to produce common, overly generic responses such as “*I don’t know*” [124, 122]. To mitigate this issue, Li et al. [77] and others have proposed methods to increase the diversity of neural network outputs. We adopt a similar approach to Li et al. [77] to generate noisier hypotheses during decoding.

5.5 Conclusion

In this chapter, we address one of the key issues for developing translation-based grammar correction systems: the need for a large corpus of parallel data. We propose synthesizing parallel data by noising clean text, where instead of applying noise based on finite context windows, we instead train a reverse model and apply noise during the beam search procedure to synthesize noisy examples that human evaluators were nearly unable to distinguish from real examples. Our experiments suggest that the proposed data synthesis technique can yields almost as strong results as when training with additional nonsynthesized data. In terms of building an AI writing assistant, we believe the data synthesis technique in this chapter is crucial for training models that suggest better edits. But, it may also be able to suggest different *types* of edits, as we’ll explore in the next chapter.

Chapter 6

Denoising as Style Transfer

Oh, hip hop is always changing.

GZA

Beyond edits to correct grammatical errors or make writing more idiomatic, we now consider edits that change the *style* of writing. Being able to write in different styles is an exciting prospect, but one that is still in its early stages. Following exciting work on style transfer in computer vision [38], neural style transfer for text has recently gained research interest as an application and testbed for syntactic and semantic understanding of sentences [81, 123, 52, 112]. Unfortunately, unlike style transfer for images, which usually requires only a single reference image, neural text style transfer often requires a large parallel corpus of sentences in the source and target style to train a neural machine translation model [128, 6].

One approach to mitigate the need for such a parallel corpus is to develop methods to disentangle stylistic attributes from semantic content, for example using adversarial classifiers [123] by predefining markers associated with stylistic attributes [81]. Such approaches can suffer from either from low quality outputs in terms of fluency and meaning preservation, or limited lexical changes instead of larger, phrase-level edits.

Given the limitations of these techniques, another approach is to try and use the vast amount of nonparallel data available across different styles and to synthesize data, for instance via backtranslation [121]. In this chapter, we take this approach

by adapting the technique described in Chapter 5. We introduce a simple method for unsupervised text style transfer that frames style transfer as a *denoising* problem in which we treat the source style as a noisy version of the target style. By further introducing hypothesis reranking techniques in the data synthesis procedure, our method—summarized in Figure 6.1—allows for rich syntactic modifications while encouraging preservation of meaning.

We evaluate our method on three distinct style transfer tasks: transfer between English varieties (American and British), formal and informal genres (news data and Internet forum data), and between lyrics of different musical genres (pop and hip hop). We use three criteria to measure the quality of outputs that have been mapped to the target style: style transfer strength, meaning preservation, and fluency. Despite the simplicity of the method, we demonstrate that it is capable of making syntactically rich suggestions. The proposed reranking technique can also be used to modulate factors such as the extent to which the style is applied or to which meaning is changed.

6.1 Method

We assume a seed corpus of parallel clean-noisy sentence pairs $(C, N) = \{(C_k, N_k)\}_{k=1}^{M_{\text{seed}}}$, as well as two non-parallel corpora $R = \{R_k\}_{k=1}^{M_R}$ and $S = \{S_k\}_{k=1}^{M_S}$ of different styles.

6.1.1 Noising

The method we use for synthesizing data for style transfer parallels the method described for grammar correction in Chapter 5. We first synthesize noisy versions of R and S . Using the seed noise corpus, we train a neural sequence transduction model to learn the mapping between clean to noisy $C \rightarrow N$. Then, we decode R and S using the noising model to synthesize the corresponding noisy versions, \tilde{R} and \tilde{S} .

- **Baseline** As a baseline, we apply the noising method described in Xie et al. [41]. This method utilizes beam search noising techniques to encourage diversity during the noising process in order to avoid copying of the inputs.

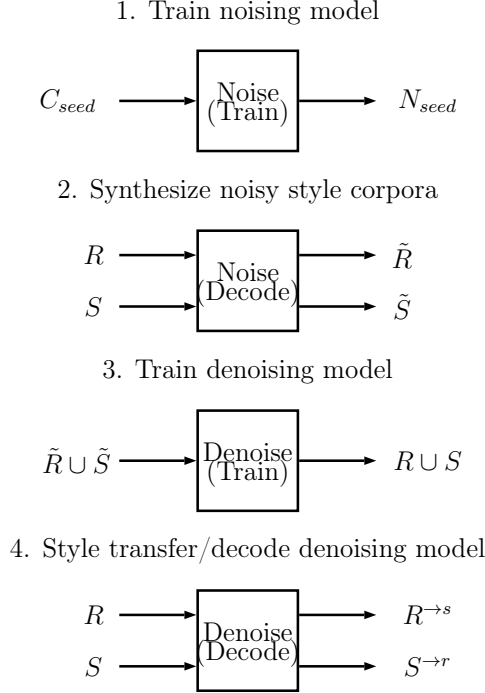


Figure 6.1: An overview of our method. We first use noising to generate synthetic parallel data in both styles, then “denoise” to transfer from one style to the other. Note the parallels to the method described in Chapter 5; we use this figure to introduce the notation for the rest of this chapter as well.

- **Style Reranking** A shortcoming of the baseline noising method is that it mimics the noise in the initial seed corpus, which may not match well with the input style. In order to produce noise that better matches the inputs that will later be fed to the denoising model, we perform reranking to bias the synthesized noisy corpora, \tilde{R} and \tilde{S} , towards the clean corpora S and R , respectively.

Consider noise synthesis for S , and denote the noising procedure for a single input as $f_{\text{noise}}(\cdot)$. We generate multiple noise hypotheses, $h_i = f_{\text{noise}}(S_k)$ and select the hypothesis closest to the alternate style R , as ranked by a language model trained on R (Fig 6.2):

$$h^* = \arg \max_i p_R(h_i)$$

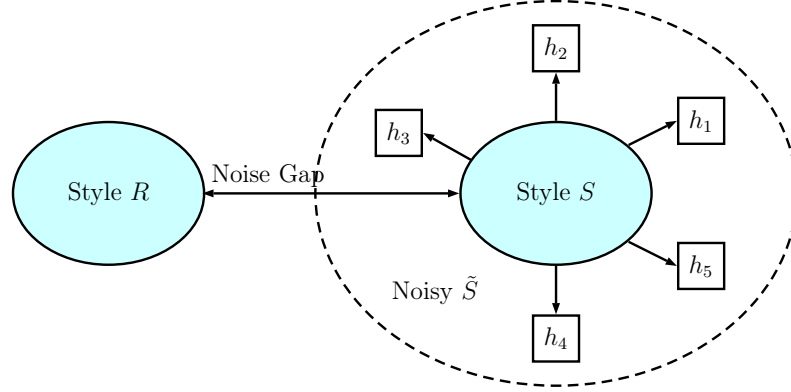


Figure 6.2: When synthesizing noisy sentences to train the denoising model $\tilde{S} \rightarrow S$, we use style reranking to choose the noisy hypothesis h closest to the style R .

- **Meaning Reranking** Similar to style reranking, we encourage meaning preservation by ranking the different noise hypotheses according to the meaning preservation score defined in Section 6.2.2.

6.1.2 Denoising

After the synthesized parallel corpus is generated, we train a denoising model between the synthesized noisy corpora and the clean counterparts. To encode style information, we prepend a start token to each noisy sentence corresponding to its style, i.e. $\tilde{R}_k = (\langle \text{style} \rangle, w_1, w_2, \dots, w_T)$.

Besides providing a simple method to specify the desired target style, this also allows us to combine the noisy-clean corpora from each of the two styles and train a single model using both corpora. This provides two benefits. First, it allows us to learn multiple styles in one model. This allows one model to perform style transfer from both $R \rightarrow S$ and $S \rightarrow R$. Second, multi-task learning often improves the performance for each of the separate tasks [88].

We then join the corpora to obtain the clean-noisy sentence pairs, $(X, \tilde{X}) = \{(X_k, \tilde{X}_k)\}_{k=1}^{M_{R+S}}$, from which we will learn our denoising model. Our denoising model

learns the probabilistic mapping $P(X|\tilde{X})$, obtaining model parameters θ^* by minimizing the usual negative log-likelihood loss function:

$$\mathcal{L}(\theta) = - \sum_{k=1}^{M_{R+S}} \log P(X_k|\tilde{X}_k; \theta)$$

For our experiments we choose the Transformer encoder-decoder model [131] with byte-pair encoding [120]. We follow the usual training procedure of minibatch gradient descent on the loss.

The trained denoising model is then applied to the source style—treated as the “noisy” corpus—with the start token of the target style to perform style transfer (Figure 6.1).

6.2 Experiments

6.2.1 Data

We evaluate our methods on three different style transfer tasks between the following corpus pairs: (1) American and British English, (2) formal news writing and informal forum writing, and (3) pop and hip hop lyrics. The first task of transferring between American and British English is primarily intended as a preliminary test for our proposed technique by demonstrating that it can capture lexical changes. The latter tasks require more sophisticated syntactic edits and form the basis of our followup analysis.

A summary of the datasets used for the three tasks is provided in Table 6.1. We use sentences from The New York Times for the American English data, sentences from the British National Corpus for the British English data, and comments from the Reddit comments dataset for informal forum data. The pop and hip hop lyrics are gathered from MetroLyrics.¹ For the parallel seed corpus used to train the noising model, we use the same dataset as in Chapter 5 [129].

¹<https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics>

Task	Dataset	Training	LM
US/UK	NYT/BNC	800K	2.5MM
Forum/News	NYT/Reddit	800K	2.5MM
Music Genres	Hip Hop/Pop	500K	400K

Table 6.1: Style transfer datasets and sizes (in number of sentences). *Training* refers to examples used to synthesize noisy sentences and train the denoising model. *LM* refers to examples used to train language models for reranking and evaluation. In addition to Training and LM, 20K examples are held out for each of the dev and test sets.

6.2.2 Evaluation

We define effective style transfer by the following criteria:

1. **Transfer strength** For a given output sentence, effective style transfer should increase the probability under the target style distribution relative to the probability of observing it under the source style distribution. We thus define transfer strength as the ratio of target-domain to source-domain shift in sentence probability. Let R be the source style inputs and $R^{\rightarrow \text{tgt}}$ be the target style outputs. Then,

$$\begin{aligned}
\text{SHIFT}_{\text{src}} &= \exp \left[\frac{1}{n} \sum_{k=1}^n \log(P(R_k^{\rightarrow \text{tgt}} | LM_{\text{src}})) \right. \\
&\quad \left. - \frac{1}{n} \sum_{k=1}^n \log(P(R_k | LM_{\text{src}})) \right] \\
\text{SHIFT}_{\text{tgt}} &= \exp \left[\frac{1}{n} \sum_{k=1}^n \log(P(R_k^{\rightarrow \text{tgt}} | LM_{\text{tgt}})) \right. \\
&\quad \left. - \frac{1}{n} \sum_{k=1}^n \log(P(R_k | LM_{\text{tgt}})) \right] \\
\text{TRANSFERSTRENGTH}_{\text{src} \rightarrow \text{tgt}} &\stackrel{\text{def}}{=} \frac{\text{SHIFT}_{\text{tgt}}}{\text{SHIFT}_{\text{src}}}
\end{aligned}$$

A positive transfer is any ratio greater than one.

2. **Meaning preservation** The target must have a similar meaning and intent to the source. To measure this, we compute the cosine similarity between embeddings r of the source and target:

$$\text{MEANINGPRESERVATION} \stackrel{\text{def}}{=} \frac{r_{\text{src}}^\top r_{\text{tgt}}}{\|r_{\text{src}}\| \|r_{\text{tgt}}\|}$$

To compute the embeddings r , we use the sentence encoder provided by the InferSent library, which has demonstrated excellent performance on a number of natural language understanding tasks [26].

3. **Fluency** The post-transfer sentence should be fluent for human readers. We use the average log probability of the sentence post-transfer with respect to a language model trained on CommonCrawl (the same as that used for rescoring our denoising models from previous chapters) as our measure of fluency.

The source and target language models are 4-gram (in the case of music lyrics) or 5-gram (in the case of other datasets) language models trained on a held-out subset of each corpus, estimated with Kneser-Ney smoothing using KenLM [48].

6.2.3 Results

Task	NYT↔BNC		NYT↔Reddit		Pop↔HipHop	
	→	←	→	←	→	←
Baseline	1.315	1.227	1.834	1.189	1.097	1.086
Style Rerank	1.359	1.274	1.992	1.254	1.110	1.072
Meaning Rerank	1.285	1.222	1.281	1.145	1.118	1.092

Table 6.2: The transfer strength for each style transfer task.

As shown in Table 6.2, we had positive style transfer on all six transfer tasks. For the task of British and American English as well as formal news writing and informal forum writing, applying style reranking during the noising process increased

Task	NYT \leftrightarrow BNC		NYT \leftrightarrow Reddit		Pop \leftrightarrow Hiphop	
	\rightarrow	\leftarrow	\rightarrow	\leftarrow	\rightarrow	\leftarrow
Baseline	92.22	92.17	96.49	91.73	96.84	97.22
Style Rerank	91.93	91.66	96.50	91.26	96.84	97.18
Meaning Rerank	94.40	93.47	98.34	94.18	97.29	97.48

Table 6.3: Meaning preservation for each style transfer task. All reported numbers scaled by 10^2 for display.

Task	NYT \leftrightarrow BNC		NYT \leftrightarrow Reddit		Pop \leftrightarrow Hiphop	
	\rightarrow	\leftarrow	\rightarrow	\leftarrow	\rightarrow	\leftarrow
Pre-Transfer	5.763	3.891	4.609	5.763	2.470	1.453
Baseline	4.016	4.012	3.920	5.506	2.112	1.429
Style Rerank	3.877	3.992	3.603	5.194	1.930	1,310
Meaning Rerank	3.874	3.743	3.808	5.395	1.915	1.284

Table 6.4: Fluency of each style transfer task. All reported numbers scaled by 10^3 for display.

the transfer strength across all four of these tasks. On the other hand, applying meaning reranking during the noising process decreased the transfer strength. For pop and hip hop lyrics, we do not observe the same pattern between the three models; however, this may be attributed to the lack of data for the language model, thereby leading to less effective style reranking. We address the possibility also of a mismatch with the initial seed corpus in Section 6.3.3.

As noted in Table 6.3, the meaning rerank method outperformed the other two models on the meaning preservation metric across all six tasks, showing that our reranking method is able to optimize for other objectives such as meaning preservation. In all six style transfer tasks in Table 6.4, fluency was highest for the baseline model among all the other models, although fluency is better in the initial corpus.

Task	Source	Target
UK to US	As the BMA’s own study of alternative therapy showed, life is not as simple as that.	As the <i>F.D.A.’s</i> own study of alternative therapy showed, life is not as simple as that.
US to UK	The Greenburgh Drug and Alcohol Force and investigators in the Westchester District Attorney’s Narcotics Initiative Program Participated in the arrest.	The <i>Royal Commission on Drug and Attache Force</i> and investigators in the Westchester District Attorney’s Initiative Program Participated in the arrest.
NYT to Reddit	The votes weren’t there.	<i>There weren’t any upvotes.</i>
Reddit to NYT	drugs are bad, mkay.	<i>The drugs are bad, Mr. McKay.</i>
Pop to Hip Hop	My money’s low	My money’s <i>on the low</i>
Hip Hop to Pop	Yo, where the hell you been?	Yo, where the hell <i>are you?</i>

Table 6.5: Qualitative examples of style transfer results for different tasks. No parallel data outside of the initial noise corpus was used. Note that the style transfer approach can generate targets with significant syntactic changes from the source. All examples shown are without reranking during data synthesis. More qualitative examples for methods using reranking can be found at the end of this chapter.

6.3 Discussion

After experimental evidence that the proposed method can produce stylistic edits, we wished to better understand the effects of the reranking methods as well as the choice of our initial seed corpus.

6.3.1 Style and Meaning Human Evaluation

While language model score is a natural measure of fluency or grammaticality, we wished to validate that the metrics for transfer strength and meaning preservation matched with human perception. We randomly selected 200 sentences for the NYT to Reddit task, then took the outputs with models trained with style reranking and meaning reranking. We then randomized the outputs such that the human evaluator

would not be given the label for which output was produced using which reranking method.

For meaning preservation, we then labeled each (randomized) pair with the sentence that seemed to have higher meaning preservation. For transfer strength, we did the same, except we allowed for a “No preference” option for cases where neither output seemed to have higher transfer strength. We chose pairwise comparisons as it seemed most robust to sometimes minor changes in output.

Computing Jaccard (intersection over union) similarity between the automatic meaning preservation metric (simply by picking the sentence with higher meaning preservation score) and the human annotations, we obtained a Jaccard similarity of 89.5. For transfer strength, Jaccard similarity was 88.5.

6.3.2 Distribution of Edits

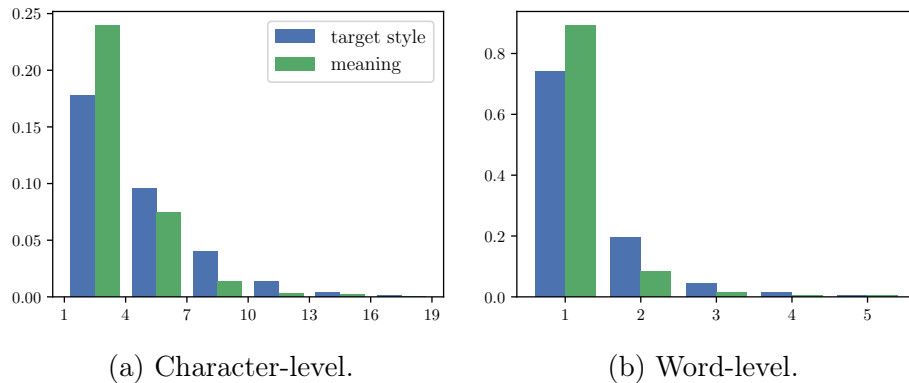


Figure 6.3: Normalized distributions of contiguous edit lengths (character and word-level) for NYT sentences mapped to Reddit style. We compare models trained on data generated with style reranking and meaning preservation reranking.

Previous work such as Li et al. [81] begins by substituting attribute phrases from one style to another. While sentiment might be characterized by specific attribute phrases, other style transfer tasks (e.g. formal and informal writing) require larger syntactic changes beyond phrase substitution. To validate that our method is capable of generating significant substitutions, we computed the contiguous lengths of edits for the NYT to Reddit transfer task. Figure 6.3 illustrates that though many edits tend to

be short, there is a tail of longer edits generated by our system. Unsurprisingly, edits generated with meaning preservation reranking skew shorter than edits generated with style reranking.

6.3.3 Limitations of Noise Corpus

NYT	686 (460)	BNC	608 (436)
Reddit	287 (215)	Pop	702 (440)
Hip hop	1239 (802)		

Table 6.6: Perplexities with (and without) OOVs for different datasets under seed corpus language model.

A key factor in the performance of our style transfer models is the method by which the initial parallel corpus is synthesized. Our method relies on an initial seed corpus of (clean, noisy) sentence pairs to bootstrap training. Yet, such a corpus is likely not ideal for the style transfer tasks we consider, as in many cases the style domains—news, music lyrics, forum posts—do not match with the seed corpus—language learner posts. We observe in Table 6.2 that more significant transfer appears to occur for the tasks involving news data, and less for music lyrics.

To examine why this might be the case, we trained a 5-gram language model on the clean portion of the initial seed corpus, corresponding to the input of the noise model. We then measured the perplexity of this language model on the different domains. Results are given in Table 6.6. The results perhaps indicate why style transfer between lyrics of different music genres proved most difficult, as there is the greatest domain mismatch between the initial seed corpus and those corpora.

6.4 Related Work

Our work is related to broader work in training neural machine translation models in low-resource settings, work examining effective methods for applying noise to text,

as well as work in style transfer.

Machine translation Much work in style transfer builds off of work in neural machine translation, in particular recent work on machine translation without parallel data using only a dictionary or aligned word embeddings [70, 4]. These approaches also use backtranslation while introducing token-level corruptions avoid the problem of copying during an initial autoencoder training phase. They additionally use an initial dictionary or embedding alignments which may be infeasible to collect for many style transfer tasks. Finally, our work also draws from work on zero-shot translation between languages given parallel corpuses with a pivot language [55].

Noise To our knowledge, there has been no prior work to formulate style transfer as a denoising task outside of using token corruptions to avoid copying between source and target. Our style transfer method borrows techniques from the field of noising and denoising to correct errors in text. We apply the noising technique in Xie et al. [44] that requires an initial noise seed corpus instead of dictionaries or aligned embeddings. Similar work for using noise to create a parallel corpus includes Ge et al. [39].

Style transfer Existing work for style transfer often takes the approach of separating *content* and *style*, for example by encoding a sentence into some latent space [14, 52, 123] and then modifying or augmenting that space towards a different style. Hu et al. [52] base their method on variational autoencoders [63], while Shen et al. [123] instead propose two constrained variants of the autoencoder. Yang et al. [142] use language models as discriminators instead of a binary classifier as they hypothesize language models provide better training signal for the generator. In the work perhaps most similar to the method we describe here, Prabhumoye et al. [112] treat style transfer as a backtranslation problem, using a pivot language to first transform the original text to another language, then encoding the translation to a latent space where they use adversarial techniques to preserve content while removing style.

However, such generative models often struggle to produce high-quality outputs.

Li et al. [81] instead approach the style transfer task by observing that there are often specific phrases that define the attribute or style of the text. Their model segments in each sentence the specific phrases associated with the source style, then use a neural network to generate the target sentence with replacement phrases associated with the target style. While they produce higher quality outputs than previous methods, this method requires manual annotation and may be more limited in capturing rich syntactic differences beyond the annotated phrases.

6.5 Conclusion

In this chapter, we describe a simple method for performing text style transfer by treating the source text as a noisy version of the desired target. Our method can generate rich edits to map inputs to the target style. We additionally propose two reranking methods during the data synthesis phase intended to encourage meaning preservation as well as modulate the strength of style transfer, then examine their effects across three varied datasets. An exciting future direction is to develop other noising methods or datasets in order to consistently encourage more syntactically complex edits.

Task	Source	Target	
		<i>Target style reranking</i>	<i>Meaning reranking</i>
NYT to Reddit	They are now relics of past campaigns, as ancient as the torchlight parade.	They are now the <i>subreddit</i> of past campaigns, as ancient as the <i>rest of the</i> parade.	They are now relics of past campaigns, as ancient as the torchlight parade.
NYT to Reddit	To judge from Mr. Levine’s rippling passagework, his left-hand tremor and right-shoulder injury early this year have not impaired his fine-motor skills.	To judge from Mr. Levine’s <i>retroll</i> , <i>his r/pics</i> , and <i>r/reddit</i> injuries early this year have not impaired his <i>to-days</i> skills.	To judge from Mr. Levine’s rippling passagework, his left-hand tremor and right-shoulder injury early this year have not impaired his fine-motor skills.
Reddit to NYT	Where are you fishing?	“Where are you fishing?”	Where are you fishing <i>for</i> ?
Reddit to NYT	i collect 21 days worth of minutely shots so i can figure out double check billing time issues for the last billing period.	<i>I collect 21 days’</i> worth of <i>substantive</i> shots so <i>I</i> can figure out double check billing time issues for the last billing period.	<i>I collect 21 days</i> worth of minutely shots so <i>I</i> can figure out double check billing time issues for the last billing period.
Pop to Hip Hop	Harken! the machines grind their teeth	Harken! the machines grind <i>they</i> teeth	Harken! the machines grind <i>they</i> teeth
Pop to Hip Hop	We are what we wear	We <i>be</i> what we wear	We are what we <i>wearing</i>
Hip Hop to Pop	Don’t even respect your a@#	<i>Never</i> respect your a@#	<i>Doesn’t</i> even respect your a@#
Hip Hop to Pop	I can’t take it, I’m going out my mind,	I can’t take it, I’m <i>leaving</i> my mind,	I can’t take it, I’m going out <i>of</i> my mind,
		<i>Meaning reranking</i>	<i>Target style reranking</i>
NYT to Reddit	<i>Drama</i> asked Lil Jon to be the host of a mixtape, and Jon did a manic series of drops throughout Gangsta Grillz No. 4.	<i>Drake</i> asked Lil Jon to be the host of a mixtape, and Jon did a manic series of drops throughout Gangsta Grillz No. 4.	<i>Draw</i> asked Lil Jon to be the host of a mixtape, and Jon did a manic series of drops throughout Gangsta Grillz No. 4.
NYT to Reddit	jokes.	<i>r/jokes</i> .	jokes....
Reddit to NYT	Still i liked it.	<i>I still</i> liked it.	<i>I still</i> liked it.
Reddit to NYT	<i>Faraday</i> has studies the properties of the island all his life he says .	<i>Mr. Faraday</i> has studies the properties of the island all his life, he says.	<i>Fewer days</i> have studied the properties of the island all his life, he says.
Pop to Hip Hop	Better worship it	Better worship it (<i>worship it</i>)	Better worship <i>them</i>
Pop to Hip Hop	So tell me how you feel baby, tell me	So tell me how you <i>feeling</i> baby, tell me	So tell me how you feel, baby, tell me
Hip Hop to Pop	Sometimes I wish Yams was never using	Sometimes I wish Yams <i>never used</i>	Sometimes I wish Yams <i>were</i> never using
Hip Hop to Pop	And i know the job move a little slow	And <i>I</i> know the job <i>moves</i> a little slow	And <i>I</i> know the job <i>moves</i> a little slow

Table 6.7: Additional qualitative examples of style transfer results for second two tasks (chosen since they involve more complex rephrasings). Again, no initial parallel data outside of the initial noise corpus was used. We divide the examples into those generated by models trained using using (i) target style reranking and (ii) meaning preservation reranking.

Chapter 7

Deployment

In language there are no licensed practitioners, but the woods are full of midwives, herbalists, colonic irrigationists, bonesetters, and general-purpose witch doctors. . .

Dwight Bolinger

In the previous chapters, we sought to develop more better-performing systems for suggesting edits to writing. This chapter describes our effort to deploy such systems in order to (1) further evaluate their effectiveness as well as to (2) examine whether there might be additional challenges we observe in the wild.

Writing is a time-consuming process where it is easy to make errors, phrase clauses poorly, or write in a style or tone inappropriate for a given audience. AI-based writing feedback has been implemented by practitioners in industry for some time [61, 93]. However, given the limitations of current AI techniques as well as the fact that deep learning-based systems are not yet broadly deployed for word processing, at present writers must often rely on friends, colleagues, or even expensive paid editors to revise and make suggestions to their writing [33].

With deep learning-based methods now state-of-the-art on benchmarks evaluating grammatical and fluency error correction [139], the question we consider is whether

systems using deep learning can be useful for everyday people who write. We engineered a software system, including a frontend interface, backend text processing system, and neural sequence transduction model, then deployed it for anyone to install and use. We named the final system “Crio” (“Creole”), as we hoped that the resulting system would allow for a mixed language between human and AI, where the human would provide some initial text, receive suggestions from the AI writing assistant, and then iteratively refine the text together.

The primary contribution of this chapter, beyond the description of the developed system, is the finding that a deep learning-based assistant for suggesting edits to real-world writing is both feasible and *useful*. This is validated through surveys early on in development followed by extensive usage of our tool by thousands of individuals. Another key finding, which to the best of our knowledge has not been described for writing assistants, is the importance of *integrating* the tool into existing user workflows. This led to several iterations of the interface we describe in this chapter. Finally, we examine the demographics of users on the system and find that a key challenge that remains for such writing assistants is factoring in other diverse aspects such as the type of language and the communication setting at hand.

7.1 Related Work

Extensive work has been done on providing automated edits to writing stretching back to the 1960s [68]. However, the majority of this work relies on count statistics, pre-defined dictionaries, or machine learning models such as support vector machines and multilayer perceptrons trained on relatively small word contexts and training sets. More recent approaches using neural machine translation on large parallel corpuses that frame the error correction task as a sequence to sequence mapping have greatly exceeded the performance of older approaches, as described in Chapters 3 and 5. However, to the best of our knowledge, no work has been published on the efficacy of these recent approaches when put into production.

Another technique for constructing a writing assistant is by crowdsourcing suggestions. Bernstein et al. [12] describe a method for crowdsourcing human suggestions

within word processors, and present crowdsourcing approaches for text shortening, spelling and grammar suggestions, and formatting and insertion of figures. Similarly, crowdsourcing has also been used for suggesting messages in conversations directly [71]. Our approach, though limited, is entirely based off of an automated machine learning system.

A significant component of the work in this chapter relates to deep learning infrastructure. Findings from industry publications on productionizing automatic speech recognition and machine translation systems influenced the system architecture described in this chapter [11, 137], in particular with optimizations made to reduce the latency of our system.

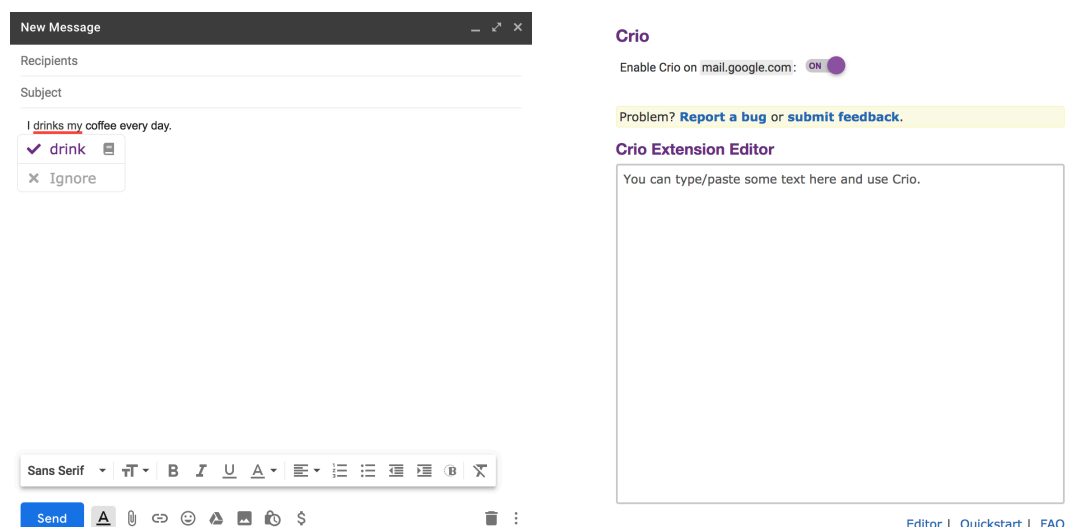
7.2 Architecture

We begin by describing the tool and its architecture, after which we describe some shortcomings and benefits of the architectural choices made.

7.2.1 Frontend

User Suggestion Display After receiving a decoded output from our model that has been provided some source input, we must then present that output to the user. In order to display edits in an unobtrusive way, we follow other word processing systems [15] and display word or phrase spans for which an edit was suggested by underlining them. A user can then hover over a particular piece of underlined text to see a suggestion, with the option to accept or reject that suggestion as a feedback signal of our suggestion quality. Figure 7.1a shows Crio providing inline suggestions for an email text input.

Channel Integrations Our primary adoption channel for Crio is through a browser extension. After installing the extension, users automatically receive suggestions on a whitelisted set of hostnames. Specifically, as they type into text inputs, the text is periodically sent to Crio’s backend server, which queries the deep learning model



(a) Crio providing inline suggestions in an email compose window. The suggestion can be accepted or rejected and appears dynamically as the user composes the sentence.

(b) Dropdown menu for hostnames that are not supported through the Crio whitelist. Users also have the option of using the editor on the Crio website.

Figure 7.1: Screenshots of the final Crio interface as a browser integration.

and sends suggestions back which are rendered to the user. Figure 7.1 illustrates the browser integration interface. We also developed an interface for longer documents (such as on Google Docs) that queries smaller portions of the document at a time.

7.2.2 Backend

The backend includes an in-memory cache, request queue, parsing library for text markup, tokenizer, and text pre- and post-processing unit. The in-memory cache records which sentences have already been processed, and, given that the majority of a given document will remain unchanged between queries, avoids re-computations for longer documents. The rest of the components prepare the text to be inputted to the neural network model, and then reform the text to be rendered again. At the core of the backend is the deep learning model server, which, given an input sentence, transduces the corresponding output sentence.

For most of our pre-processing we rely on Stanford CoreNLP¹ for extensive options and better handling of sentence and word boundaries than other available libraries. To handle unexpected characters (e.g. from other languages), we explicitly filter such examples from being queried through the deep learning server.

7.2.3 Deep Learning Server

Our choice of architecture has evolved with the evolution of the state-of-the-art architectures for the tasks described in previous chapters. Our ultimate architecture for mapping from noisy source sentences to clean targets is the encoder-decoder version of the Transformer architecture described in [131]. For latency reasons, the models are served on NVIDIA K80 GPUs with 64 vCPUs and 700 GB of host memory.

Although speed of decoding is not a huge concern during training, it is a concern for AI writing assistants, where real-time decoding is often a requirement. Beyond gains from using highly parallelized hardware such as GPUs or from using libraries with optimized matrix-vector operations, we use byte-pair encoding [120] tokenization—to trade off size and flexibility of the vocabulary (softmax) and number of decoder timesteps—and batch multiple examples together. We also perform as much computation as possible within the compiled computation graph, meaning no rescoring until after the final list of hypotheses are determined.

7.3 Results & Discussion

In the following subsections we describe the three key findings from deploying the architecture we describe above.

7.3.1 Importance of Integrations

Our first finding is the importance of integrating Crio into existing user workflows. This was a hard-earned lesson. The first iteration of Crio was a standalone website in

¹<https://github.com/stanfordnlp/CoreNLP>



Figure 7.2: Early version of Crio which was a standalone website. Users were provided with a basic text editor as well as a saved list of entries they had previously composed. Despite a similar interface to that described in Section 7.2.1, users were not retained.

which users could write in our own text input environment, as shown in Figure 7.2. Login was not required.

A standalone website was appealing as we could control the editor or text input, whereas otherwise we would have to adapt to the text input on other services which can greatly differ and completely change without our knowledge.

To obtain initial feedback, 29 Amazon Mechanical Turk workers were hired to visit the website, obtain corrections for 5 different sentences, and post the received suggestions to a public listing. They were then asked to fill out a survey with questions about the service.

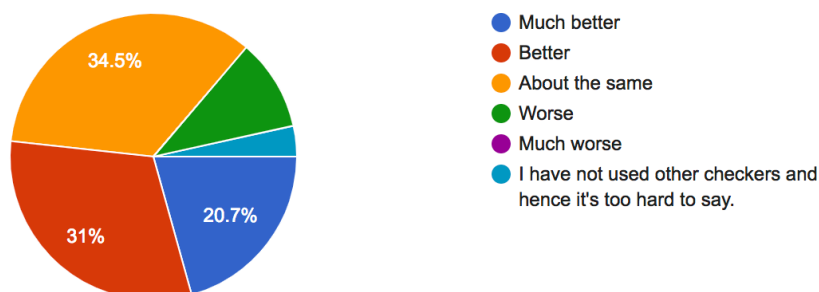


Figure 7.3: Summary of survey question for users of early Crio system. The question was, “How would you rate the automatic grammar suggestions by [Crio] compared to other grammar checkers you have used?”.

As shown in Figure 7.3, users generally found Crio’s suggestions to be at least as good as existing services they used. However, none of the users were retained—i.e., they did not continue to use the standalone website. In terms of the Crio software, we suspect the reasons for this are twofold:

1. In order to retain a user for the standalone service, the user must revisit the standalone service, which they may not remember to do. On the other hand, in order to retain a user for the browser integration, we only need that the user not *uninstall* the integration. Over a typical month, less than 10% of users uninstall. Thus, a standalone user is lost unless they take an action, while an integration user is kept unless they go out of their way to not continue using the service.
2. The integration provides a consistent reminder of the service by giving suggestions on services used almost daily by users, such as services for email and blog writing.

These reasons are supported by the steady usage of Crio by new users after we switched to a browser integration, as well as by the survey responses.

The other possible factor unrelated to the software is that the users selected through Turk are very different from the users who eventually discovered and used Crio. However, if we examine the number of unique users over a one month snapshot who accept an edit on any of the other hostnames—as opposed to the Crio editor on `crio.stanford.edu`, our closest equivalent to the original standalone service—we find that only $\approx 20\%$ of our total accepts originate from the Crio editor as opposed to other hostnames. Further, this may be a significant overestimate as the Crio editor is the very first text input that a user who installs Crio encounters as part of onboarding. Thus, an integration apart from a standalone service clearly remains critical for such a writing assistant.

7.3.2 Deep Learning-Based Suggestions Appear Useful

The second key finding and the primary hypothesis we hoped to determine from deploying Crio is that suggestions from our deep learning-based model appear to be

useful. Over the several months over which Crio deployed, we saw thousands of users. As of this writing, roughly 7000 users have installed each of the browser and docs integrations (15K combined), and over 3000 unique users accept at least one edit per week. Users from over 100 countries have installed and used Crio. Table 7.1 summarizes the usage statistics.

Metric	Value
Combined installations	15K
Text inputs queried/day	32K
Total accepts/day	16K
Total rejects/day	2.8K
Most accepts by user in a day	Varies, 250-900 typical
Countries	113
Supported hostnames	12

Table 7.1: Crio usage statistics.

Of particular interest is a subset of users who accept dozens and sometimes even hundreds of edits per day. We find these are typically users who are composing longer documents, such as reports, technical manuals, or fiction novels, for whom accurate writing is important. One final indicator is that users will often accept edits, but infrequently reject edits—in practice, the ratio of accepts to rejects is about 5:1. Although this is confounded by users not needing to reject edits before sending an email or posting a writing entry, it nevertheless suggests that the deep learning model is reasonably effective.

7.3.3 Diversity of Users and Writing

The last finding from deployment is the diverse set of users and settings in which AI writing assistants are used and must handle. As given in Table 7.1, users from 113 countries used Crio, and we saw usage from the various supported hostnames as well as other hostnames in which users simply used the dropdown text input.

We hypothesized that, given the different types of users and writing channels, a key challenge in building a better AI writing assistant would be the ability to handle

dissimilar users and domains (excluding the clear challenge of different languages entirely). To test this hypothesis, we use analytics collected from Crio’s deployment to test whether:

1. There were differences in suggestion quality for users from countries with different English varieties.
2. There were differences in suggestion quality between suggestions across different websites of varying formality.

To evaluate suggestion quality, we use the metric

$$\text{suggestion quality} = \frac{\# \text{ accepts}}{\# \text{ suggestions}} \times 100,$$

a rough user-determined estimate of the precision of suggestions. Due to our observation that edits are rejected with much lower frequency than they are accepted (due to it not being necessary to reject edits to achieve any user outcome besides removing the underline), as well as our observation of user fatigue due to repeated suggestions, we do not use rejects as part of the evaluation metric.

Varieties of English

As a first evaluation of the performance of Crio for different users, we considered users requesting suggestions from different English-speaking countries. Our intention is to determine whether Crio may have lower suggestion quality for different language varieties, due to the majority of its training data being formal U.S. English.

The results for users from countries with different varieties of English is given in Table [7.2](#). We observe that the suggestion quality is highest for in-domain queries from the U.S., as expected, and that suggestion quality is consistently lower for countries where other variants of English are used. This suggests models that adapt to each of these different settings could improve the performance of these AI writing systems.

Country	Suggestion Quality
United States	6.96
United Kingdom	4.87
Canada	6.18
India	5.57

Table 7.2: Summary of suggestion quality for countries with different English varieties over one month period. The smallest sample size of suggestions made is 210K (UK).

Hostname	Suggestion Quality
mail.google.com	5.89
facebook.com	2.45
outlook.office.com	5.93
reddit.com	3.00

Table 7.3: Summary of suggestion quality for different web services over one month period. The smallest sample size of suggestions made is 20.2K (`reddit.com`).

Varieties in Setting

As another examination of suggestion quality, we consider different websites of with varying formality and use. We consider two email services (Gmail and Microsoft Outlook) as well as an informal forum setting (Reddit) and a social network (Facebook). As shown in Table [7.3](#), suggestion quality is significantly lower for more casual websites. Although one may argue that users inherently care less about their writing on such websites, AI writing assistants should also understand this and be able to provide suggestions that are helpful specifically for casual writing as well.

7.4 Conclusion

We deployed Crio, an AI writing assistant for any user to install and use. From deployment we made findings along different fronts. On the HCI front, we found that (1) integrations are crucial to retain and encourage use of such writing assistants

and (2) that deep learning-based suggestions can be useful. On the data and model development front, we found that a key remaining challenge is developing systems that maintain the same suggestion quality across different varieties of language and different writing settings. An exciting direction of future work is closing the loop and developing adaptive writing assistants to see if this significantly improves their performance and use.

Chapter 8

Conclusion

In this thesis, we address challenges towards developing a deep learning-based AI writing assistant. We began by tackling the problem of noise in text, both through editing text containing errors as well as by better understanding the effects of noise during sequence modeling. Through better understanding of the insufficiency of naive token deletions and substitutions, we then developed a better noising method in order to synthesize additional data to train denoising models. Finally, we developed and deployed an AI writing assistant, Crio, that integrates into user’s browsers and document editors to suggest edits.

At the outset, the focus of this work was on developing AI systems that are more robust to unexpected inputs and to better understand their behavior under noisy conditions. Upon further investigation, we discovered that another view of the problem is that of insufficient data, and hence we developed data synthesis techniques. After deployment, we found yet a third view—that the next challenge is coping with the diverse set of users and settings to which these systems must adapt.

8.1 Other Remaining Challenges

Besides the problem of adapting to different users and settings, there are also other challenges we did not consider in this thesis:

- **Multiple phrasings** Beyond correctness, there are often multiple possible suggestions to improve a sentence. Crio suggests only a single edit per span; however, the ability to give multiple (non-redundant) suggestions could significantly improve the recall of these systems. This would involve classifying when multiple suggestions might be relevant as well.
- **Higher-level feedback** Another exciting direction is to give higher-level suggestions at the paragraph or document level. Such methods would likely involve less language modeling-based objectives and involve the collection of more structured annotations. Other related tasks include automatic shortening of texts [12].
- **More controllability** As a motivating toy example, consider the two misspellings “ballon” and “ballonn” (instead of “balloon”). An edit distance-based correction algorithm would never correct the second misspelling without correcting the first. Yet, we have observed neural systems make non-transitive suggestions such as this. Such intuitions also apply to other cases, for example where the neural network replaces a clear proper noun for another. While full controllability of these systems appears a long way off, such common sense scenarios may be handled in the near term.

8.2 Final Remarks

Consider the spectrum of possible suggestions that an AI assistant could provide. At one extreme we have the simplest of suggestions, perhaps spelling corrections or the addition of punctuation here and there. Then, proceeding along the spectrum, we suggest local edits, the rephrasings of clauses. Much further along, given some previous text or conversation history, an AI assistant could suggest entire sentences and replies, with the option to select one of many options. While written language remains the medium of interaction—and not a higher throughput channel such as a brain-machine interface—this seems like the furthest point along the spectrum. We hope the methods and findings in this thesis aid as a step towards that goal.

Bibliography

- [1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.
- [2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.
- [3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.
- [4] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. In *Proceedings of the Sixth International Conference on Learning Representations*, April 2018.
- [5] AiTi Aw, Min Zhang, Juan Xiao, and Jian Su. A phrase-based statistical model for sms text normalization. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 33–40. Association for Computational Linguistics, 2006.
- [6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2014.

- [7] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *ICLR*, 2017.
- [8] Doug Beeferman, Adam Berger, and John Lafferty. Cyberpunc: A lightweight punctuation annotation system for speech. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 2, pages 689–692. IEEE, 1998.
- [9] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. *ICLR*, 2018.
- [10] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Neural Information Processing Systems (NIPS)*, 2015.
- [11] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. In *Journal Of Machine Learning Research*, 2003.
- [12] Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. Soyent: a word processor with a crowd inside. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*, pages 313–322. ACM, 2010.
- [13] John Bitchener, Stuart Young, and Denise Cameron. The effect of different types of corrective feedback on esl student writing. *Journal of second language writing*, 14(3):191–205, 2005.
- [14] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *CoNLL*, 2016.

- [15] Eric Brill and Robert C Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Association for Computational Linguistics, 2000.
- [16] Chris Brockett, William B Dolan, and Michael Gamon. Correcting esl errors using phrasal smt techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 249–256. Association for Computational Linguistics, 2006.
- [17] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 1992.
- [18] Christopher Bryant and Hwee Tou Ng. How far are we from fully automatic high quality grammatical error correction? In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 697–707, 2015.
- [19] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE, 2016.
- [20] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Association for Computational Linguistics (ACL)*, 1996.
- [21] Stanley F Chen, Douglas Beeferman, and Roni Rosenfeld. Evaluation metrics for language models. 1998.
- [22] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau,

- Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP*, 2014.
- [23] Martin Chodorow, Markus Dickinson, Ross Israel, and Joel Tetreault. Problems in evaluating grammatical error detection systems. *Proceedings of COLING 2012*, pages 611–628, 2012.
- [24] Shamil Chollampatt and Hwee Tou Ng. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, February 2018.
- [25] Chenhui Chu, Raj Dabre, and Sadao Kurohashi. An empirical comparison of simple domain adaptation methods for neural machine translation. *ANLP*, 2017.
- [26] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1070. URL <http://aclweb.org/anthology/D17-1070>.
- [27] John M Conroy and Hoa Trang Dang. Mind the gap: Dangers of divorcing evaluations of summary content from linguistic quality. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, 2008.
- [28] Daniel Dahlmeier and Hwee Tou Ng. A beam-search decoder for grammatical error correction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 568–578. Association for Computational Linguistics, 2012.
- [29] Daniel Dahlmeier and Hwee Tou Ng. Better evaluation for grammatical error correction. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2012.

- [30] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pages 3061–3069, 2015.
- [31] Hal Daume III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263. Association for Computational Linguistics, 2007. URL <http://aclweb.org/anthology/P07-1033>.
- [32] Li Deng, Alex Acero, Mike Plumpe, and Xuedong Huang. Large-vocabulary speech recognition under adverse acoustic environments. In *ICSLP*, 2000.
- [33] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 107–114. ACM, 1992.
- [34] Mariano Felice. Artificial error generation for translation-based grammatical error correction. Technical report, University of Cambridge, Computer Laboratory, 2016.
- [35] Mariano Felice and Zheng Yuan. Generating artificial errors for grammatical error correction. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 116–126, 2014.
- [36] Mariano Felice, Zheng Yuan, Øistein E. Andersen, Helen Yannakoudakis, and Ekaterina Kochmar. Grammatical error correction using hybrid systems and type filtering. In *In Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 15–24, 2014.
- [37] Yarin Gal. A theoretically grounded application of dropout in recurrent neural networks. In *NIPS*, 2016.
- [38] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

- [39] Tao Ge, Furu Wei, and Ming Zhou. Fluency boost learning and inference for neural grammatical error correction. *Association for Computational Linguistics*, 2018.
- [40] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional Sequence to Sequence Learning. In *Proc. of ICML*, 2017.
- [41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [42] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. In *ICLR*, 2017.
- [43] A. Graves and N. Jaitly. Towards End-to-End Speech Recognition with Recurrent Neural Networks. In *ICML*, 2014.
- [44] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- [45] Na-Rae Han, Martin Chodorow, and Claudia Leacock. Detecting errors in english article usage by non-native speakers. *Natural Language Engineering*, 12(2):115–129, 2006.
- [46] Na-Rae Han, Joel R Tetreault, Soo-Hwa Lee, and Jin-Young Ha. Using an error-annotated learner corpus to develop an esl/efl error correction system. In *LREC*, 2010.
- [47] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

- [48] K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn. Scalable modified Kneser-Ney language model estimation. In *ACL-HLT*, pages 690–696, Sofia, Bulgaria, 2013.
- [49] Kenneth Heafield. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197. Association for Computational Linguistics, 2011.
- [50] Kenneth Heafield and Alon Lavie. Cmu multi-engine machine translation for wmt 2010. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 301–306. Association for Computational Linguistics, 2010.
- [51] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.
- [52] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Toward controlled generation of text. In *ICML*, 2017.
- [53] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Association for Computational Linguistics (ACL)*, 2015.
- [54] Jianshu Ji, Qinlong Wang, Kristina Toutanova, Yongen Gong, Steven Truong, and Jianfeng Gao. A nested attention neural hybrid model for grammatical error correction. In *ACL*, 2017.
- [55] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google’s multilingual neural machine translation system: enabling zero-shot translation. *arXiv preprint arXiv:1611.04558*, 2016.
- [56] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

- [57] Marcin Junczys-Dowmunt and Roman Grundkiewicz. The amu system in the conll-2014 shared task: Grammatical error correction by data-intensive and feature-rich statistical machine translation. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 25–33, 2014.
- [58] Marcin Junczys-Dowmunt and Roman Grundkiewicz. Phrase-based machine translation is state-of-the-art for automatic grammatical error correction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1546–1556. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1161. URL <http://aclweb.org/anthology/D16-1161>.
- [59] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.
- [60] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.
- [61] Mark D Kernighan, Kenneth W Church, and William A Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics-Volume 2*, pages 205–210. Association for Computational Linguistics, 1990.
- [62] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
- [63] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.
- [64] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, 2015.

- [65] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39. Association for Computational Linguistics, 2017. doi: 10.18653/v1/W17-3204. URL <http://aclweb.org/anthology/W17-3204>.
- [66] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.
- [67] David Krueger and Roland Memisevic. Regularizing rnns by stabilizing activations. In *ICLR*, 2016.
- [68] Karen Kukich. Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)*, 24(4):377–439, 1992.
- [69] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1378–1387, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <http://proceedings.mlr.press/v48/kumar16.html>.
- [70] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc’Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations (ICLR)*, 2018.
- [71] Walter S Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F Allen, and Jeffrey P Bigham. Chorus: a crowd-powered conversational assistant. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 151–162. ACM, 2013.
- [72] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

- [73] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [74] John Lee and Stephanie Seneff. Automatic grammar correction for second-language learners. In *Ninth International Conference on Spoken Language Processing*, 2006.
- [75] John Lee and Stephanie Seneff. Correcting misuse of verb forms. *Proceedings of ACL-08: HLT*, pages 174–182, 2008.
- [76] Kyusong Lee and Gary Geunbae Lee. Postech grammatical error correction system in the conll-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 65–73, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/W/W14/W14-1709>.
- [77] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119. Association for Computational Linguistics, 2016. doi: 10.18653/v1/N16-1014. URL <http://aclweb.org/anthology/N16-1014>.
- [78] Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.
- [79] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1127. URL <http://aclweb.org/anthology/D16-1127>.
- [80] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation. In *Proceedings*

- of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 2157–2169. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1230. URL <http://aclweb.org/anthology/D17-1230>.
- [81] Juncen Li, Robin Jia, He He, and Percy Liang. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1865–1874. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-1169. URL <http://aclweb.org/anthology/N18-1169>.
- [82] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8, 2004.
- [83] Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W Black. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*, 2015.
- [84] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016. URL <http://aclweb.org/anthology/Q16-1037>.
- [85] Chia-Wei Liu, Ryan Lowe, Iulian Serban, Mike Noseworthy, Laurent Charlin, and Joelle Pineau. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2122–2132. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1230. URL <http://aclweb.org/anthology/D16-1230>.
- [86] Minh-Thang Luong and Christopher D Manning. Stanford neural machine translation systems for spoken language domains. In *Proceedings of the International Workshop on Spoken Language Translation*, 2015.

- [87] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2015.
- [88] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. In *ICLR*, 2016.
- [89] Thang Luong, Ilya Sutskever, Quoc Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 11–19. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-1002. URL <http://aclweb.org/anthology/P15-1002>.
- [90] Andrew L Maas, Quoc V Le, Tyler M O’Neil, Oriol Vinyals, Patrick Nguyen, and Andrew Y Ng. Recurrent neural networks for noise reduction in robust asr. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.
- [91] Maas and Xie, Dan Jurafsky, and Andrew Y. Ng. Lexicon-free conversational speech recognition with neural networks. In *Proceedings the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2015.
- [92] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, 2014.
- [93] Eric Mays, Fred J Damerau, and Robert L Mercer. Context based spelling correction. *Information Processing & Management*, 27(5):517–522, 1991.
- [94] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

- [95] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048, 2010.
- [96] Tomáš Mikolov. *Statistical language models based on neural networks*. PhD thesis, PhD thesis, Brno University of Technology. 2012.[PDF], 2012.
- [97] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 2013.
- [98] Tomoya Mizumoto, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. Mining revision log of language learning sns for automated japanese error correction of second language learners. In *International Joint Conference on Natural Language Processing (IJCNLP)*, 2011.
- [99] Tomoya Mizumoto, Yuta Hayashibe, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. The effect of learner corpus size in grammatical error correction of esl writings. *Proceedings of COLING 2012: Posters*, pages 863–872, 2012.
- [100] Tomoya Mizumoto, Yuta Hayashibe, Mamoru Komachi, Masaaki Nagata, and Yuji Matsuomto. The effect of learner corpus size in grammatical error correction of ESL writings. In *International Conference on Computational Linguistics*, 2012.
- [101] Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. Jfleg: A fluency corpus and benchmark for grammatical error correction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 229–234, Valencia, Spain, April 2017. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/E17-2037>.
- [102] Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. The CoNLL-2013 shared task on grammatical error correction. In

- Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task (CoNLL-2013 Shared Task)*, 2013.
- [103] 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 *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, 2014.
- [104] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *International Conference on Machine learning (ICML)*, 2005.
- [105] Jekaterina Novikova, Ondřej Dušek, Amanda Cercas Curry, and Verena Rieser. Why we need new evaluation metrics for nlg. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2241–2252. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1238. URL <http://aclweb.org/anthology/D17-1238>.
- [106] Bureau of Labor Statistics. American time use survey 2017 results. 2017. URL <https://www.bls.gov/news.release/pdf/atus.pdf>.
- [107] Chris Olah and Shan Carter. Attention and augmented recurrent neural networks. *Distill*, 2016. doi: 10.23915/distill.00001. URL <http://distill.pub/2016/augmented-rnns>.
- [108] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.
- [109] 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.

- [110] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
- [111] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, 2014.
- [112] Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. Style transfer through back-translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 866–876. Association for Computational Linguistics, 2018. URL <http://aclweb.org/anthology/P18-1080>.
- [113] Marek Rei, Mariano Felice, Zheng Yuan, and Ted Briscoe. Artificial error generation with machine translation and syntactic patterns. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 287–292. Association for Computational Linguistics, 2017. doi: 10.18653/v1/W17-5032. URL <http://aclweb.org/anthology/W17-5032>.
- [114] Alla Rozovskaya and Dan Roth. Generating confusion sets for context-sensitive error correction. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 961–970. Association for Computational Linguistics, 2010.
- [115] 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.
- [116] Alla Rozovskaya, Mark Sammons, and Dan Roth. The ui system in the hoo 2012 shared task on error correction. In *Proceedings of the Seventh Workshop*

- on Building Educational Applications Using NLP*, pages 272–280. Association for Computational Linguistics, 2012.
- [117] Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, Dan Roth, and Nizar Habash. The illinois-columbia system in the conll-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 34–42, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/W14/W14-1704>.
- [118] Keisuke Sakaguchi, Matt Post, and Benjamin Van Durme. Grammatical error correction with neural reinforcement learning. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 366–372. Asian Federation of Natural Language Processing, 2017. URL <http://aclweb.org/anthology/I17-2062>.
- [119] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1757–1766. The COLING 2016 Organizing Committee, 2016. URL <http://aclweb.org/anthology/C16-1165>.
- [120] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.
- [121] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1009. URL <http://aclweb.org/anthology/P16-1009>.
- [122] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, 2016.

- [123] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems*, pages 6830–6841, 2017.
- [124] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 196–205. Association for Computational Linguistics, 2015. doi: 10.3115/v1/N15-1020. URL <http://aclweb.org/anthology/N15-1020>.
- [125] 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*, 2014.
- [126] Raymond Hendy Susanto. Systems combination for grammatical error correction. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
- [127] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems (NIPS)*, 2014.
- [128] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
- [129] Toshikazu Tajiri, Mamoru Komachi, and Yuji Matsumoto. Tense and aspect error correction for esl learners using global context. In *Association for Computational Linguistics: Short Papers*, 2012.
- [130] INC. THE RADICATI GROUP. Email statistics report, 2017-2021. 2017. URL <https://www.radicati.com/wp/wp-content/uploads/2017/01/Email-Statistics-Report-2017-2021-Executive-Summary.pdf>.

- [131] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.
- [132] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.
- [133] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015.
- [134] S. Wager, W. Fithian, S. I. Wang, and P. Liang. Altitude training: Strong bounds for single-layer dropout. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.
- [135] Stefan Wager, William Fithian, and Percy Liang. Data augmentation via levy processes. *arXiv preprint arXiv:1603.06340*, 2016.
- [136] Sida I Wang, Mengqiu Wang, Stefan Wager, Percy Liang, and Christopher D Manning. Feature noising for log-linear structured prediction. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.
- [137] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [138] Ziang Xie. Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534*, 2017.
- [139] Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y Ng. Neural language correction with character-based attention. *arXiv preprint arXiv:1603.09727*, 2016.

- [140] Ziang Xie, Sida I Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y Ng. Data noising as smoothing in neural network language models. In *ICLR*, 2017.
- [141] Ziang Xie, Guillaume Genthial, Stanley Xie, Andrew Ng, and Dan Jurafsky. Noising and denoising natural language: Diverse backtranslation for grammar correction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 619–628, 2018.
- [142] Zichao Yang, Zhiting Hu, Chris Dyer, Eric P. Xing, and Taylor Berg-Kirkpatrick. Unsupervised text style transfer using language models as discriminators. In *NIPS*, 2018.
- [143] Zheng Yuan and Ted Briscoe. Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386, 2016.
- [144] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.
- [145] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.