

ABSTRACT

LEEMAN-MUNK, SAMUEL PAUL Morphosyntactic Neural Analysis for Generalized Lexical Normalization. (Under the direction of James Lester II.)

The phenomenal growth of social media, web forums, and online reviews has spurred a growing interest in automated analysis of user-generated text. At the same time, a proliferation of voice recordings and efforts to archive culture heritage documents are fueling demand for effective automatic speech recognition (ASR) and optical character recognition (OCR). These sources of text all have two qualities in common: they are high in volume, and they frequently diverge from standard language in their surface forms, making them difficult to analyze using conventional methods.

To address these challenges, we either need to update our analysis methods to be robust to noisy text, or we need to design a technique to convert such text into a predetermined standard form, or “normalize” it. This document introduces an instance of the latter approach. Many techniques have been proposed to normalize ASR, OCR, and Twitter data, but they have always been treated as separate tasks despite having much in common. To our knowledge, the work presented here is the first to unite these tasks under a single umbrella task of generalized lexical normalization and develop an approach to this task based on deep learning.

We introduce two architectures for this purpose. The first uses a simple feed-forward neural network to perform Twitter normalization. This approach is context-insensitive and achieved third place in the Lexical Normalization of English Tweets Challenge conducted with the ACL Workshop on Noisy User Text at the 2015 Annual Meeting of the Association for Computational Linguistics. Our second architecture is an extension of the first that, using concepts from neural machine translation, adds a gated bidirectional recurrent neural network to use the context in which a word appears as well as the characters in the word itself to normalize both Twitter and other sources of noisy text.

We evaluate this second architecture on optical character recognition post-processing, automatic speech recognition post-processing, and Twitter text normalization. In comparison with specialized tools for OCR postprocessing and Twitter normalization, we find that our model performs comparably on each of these tasks to the competing model specialized for it and significantly outperforms the model specialized for the other task. This indicates the ability for our model to learn to normalize different types of noise from data, and suggests that it could

similarly learn to be effective on other unseen types of noise without the need for expensive feature engineering.

© Copyright 2016 by Samuel Paul Leeman-Munk
All Rights Reserved

Morphosyntactic Neural Analysis for Generalized Lexical Normalization

by
Samuel Paul Leeman-Munk

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2016

APPROVED BY:

James Cox

Bradford Mott

Jonathan Rowe

Christopher Healey

James Lester II
Chair of Advisory Committee

DEDICATION

Dedicated to my parents Tom Munk and Jennifer Leeman, without whom none of this would have ever been possible.

BIOGRAPHY

Sam was born in New York City in October of 1988. After spending the first two years of his life in New York, he moved with his family to Cary, North Carolina, where he spent his early childhood. By the time Sam was of middle school age, his family had moved to Chapel Hill where he was placed in the excellent public education system. Here he had the opportunity to study Japanese and Latin along with advanced calculus and, equally critical, compulsory art and writing curricula.

Sam briefly left the state-run school system when he earned a bachelor's degree in computer science with a minor in Japanese language and linguistics from Earlham College, a small Quaker college in Indiana in 2010. At Earlham College, Sam worked under his professor Charles F. Peck developing multidimensional benchmarking systems for high performance cluster computing systems and helping to design curricula for and teach parallel computing workshops at such revered public institutions as The University of Illinois Urbana-Champaign. During his summers away from Earlham College, Sam interned at the National Science Foundation-funded Shodor Education Foundation in Durham, North Carolina where he developed educational software and ran interactive classes teaching middle schoolers and high schoolers from a range of economic and racial backgrounds computational science. He stayed on at Shodor an additional year after graduation from Earlham before he entered graduate school at North Carolina State University.

As another example of excellence in state-run schools, North Carolina State University's graduate program gave Sam the opportunity to study advanced artificial intelligence, machine learning, and natural language processing. While working with NCSU's Center for Educational Informatics, Sam worked with The SAS Institute of Cary on Deep Learning and its application to text analytics, receiving a fellowship from the organization that has funded the last year of his dissertation work.

ACKNOWLEDGEMENTS

I would like to thank my advisor James Lester and Senior Research Scientist Bradford Mott for their guidance throughout this work. This dissertation would not have been possible without their support. Their remarkable ability to find time in their busy schedules to provide guidance has been central to my academic success. Additionally, I am grateful for the insightful feedback and advice of the members of my committee. Along with an infectious enthusiasm for research, James Cox provided technical feedback in weekly meetings that helped keep me on productive research paths. Jon Rowe provided elaborately detailed conceptual feedback on each document that I presented to him and discussed statistical techniques with me at length until we could both agree that approximate randomization was the solution to most if not all statistical woes. Christopher Healey added a much appreciated levity to meetings and made a point of clarifying my hypotheses, particularly with regards to what “competitive” means when comparing models and to what I should and should not be applying claims of significance, two clarifications that eased for me the famous ambiguity of the dissertation process.

I thank my colleagues in the Center for Educational Informatics and the IntelliMedia Group for their help and support over the course of my degree. Individually I would like to thank Alok Baikadi, Megan Frankosky, Wookhee Min, Andy Smith, and Robert Taylor. I would also like to thank members of the various wider research communities in which I have participated and with whom I have had the opportunity to speak or who have otherwise had impacts on my research, Rodney Neilsen, Art Graesser, Valerie Schute, and Myroslava Dzikovska. I would finally like to thank the individuals who helped to shape my career before graduate school: Dr. Charlie Peck of Earlham College, Dr. Robert Panoff of the Shodor Education Foundation, and Mr. Keith Cooper of Chapel Hill High School.

There are many individuals in the department and university who have helped me achieve my goals. I would like to specifically thank Annie Anton, Kristy Boyer, Kathy Luca, Nagiza Samatova, and Andrew Sleeth for the particular help they offered me as I made my way through my graduate career. I thank everyone at the university for their work supporting this public institution that enriches so many young academics, for the benefit of both the students

themselves, the local private industries that get to take advantage of the incredible talent produced by our state university system, and the state as a whole. Let no one say that education at any level is not a profession to be honored.

I would also like to thank the many people at the SAS institute for their assistance in developing my understanding of deep learning for text analytics and their enthusiasm for the work that we did during NCSU and SAS's collaboration. In particular, I would like to name Larry Lewis and Patrick Hall. It has truly been a pleasure working with this group. I would also like to thank Ning Jin for making a model that I could reasonably refer to as the state-of-the-art in Twitter normalization, despite having to be convinced even to submit it for consideration in W-NUT, and Saratendu Sethi for helping to make sure that I got the hardware I needed to do this work.

Let me thank my family and friends for their unflagging support over the five years that I have been in graduate school. It was with their help that I developed the rigorous dedication to completing the task at hand, rudimentary grasp of basic social skills, and humble optimism required to succeed in a demanding graduate program. I would like to name in particular my parents Tom Munk and Jennifer Leeman, and my friend and former roommate Greg Euchner.

I would also like to especially thank my partner, Alice Ann Broadhead, for her contributions. Above and beyond the typical emotional support, which she indeed did provide in no small amount, Alice used her own graduate school experience to give direct assistance that catapulted me ahead at times when I otherwise would have stalled. Thank you, Alice.

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES.....	xi
CHAPTER 1 INTRODUCTION	1
1.1 THE UMBRELLA TASK OF GENERALIZED LEXICAL NORMALIZATION	2
OPTICAL CHARACTER RECOGNITION POST-PROCESSING.....	6
AUTOMATIC SPEECH RECOGNITION POST-PROCESSING	7
TWITTER NORMALIZATION	8
1.2 CHALLENGES.....	9
VARIETY OF NOISE	9
ABSENCE OF AN ENCOMPASSING STANDARD FORM.....	11
SCARCITY OF LABELED TRAINING DATA.....	11
1.3 RESEARCH QUESTION AND HYPOTHESES	12
1.4 CONTRIBUTIONS.....	15
1.5 ORGANIZATION	16
CHAPTER 2 RELATED WORK: LEXICAL NORMALIZATION	17
2.1 NOISE MODELS.....	18
2.2 LANGUAGE MODELS	21
2.3 THE W-NUT LEXICAL NORMALIZATION FOR ENGLISH TWEETS CHALLENGE.....	24
2.4 MACHINE TRANSLATION FOR LEXICAL NORMALIZATION	25
2.5 GENERAL MODELS OF LEXICAL NORMALIZATION	25
2.6 NOVELTY OF THIS WORK	26
CHAPTER 3 FEED-FORWARD NEURAL NETWORKS FOR THE NORMALIZATION OF TWITTER DATA WITHOUT CONTEXT	29
3.1 A BRIEF OVERVIEW OF DEEP NEURAL NETWORKS FOR TEXT ANALYTICS.....	30
DEEP LEARNING FUNDAMENTALS	31
SOFTMAX AND NEGATIVE LOG LIKELIHOOD	33

PROJECTION LAYERS	34
RECTIFIED LINEAR UNITS.....	35
3.2 THE W-NUT LEXICAL NORMALIZATION FOR ENGLISH TWEETS CHALLENGE.....	36
3.3 ARCHITECTURE AND COMPONENTS	36
THE RECONSTRUCTOR.....	37
THE FLAGGER.....	40
THE CONFORMER.....	41
3.4 SETTINGS AND EVALUATION	41
3.5 RESULTS AND DISCUSSION.....	43
3.6 NOTES ON THE ABSENCE OF FEATURE ENGINEERING	47
3.7 SUMMARY	48
CHAPTER 4 RELATED WORK: NEURAL MACHINE TRANSLATION.....	49
4.1 ADVANCED NEURAL NETWORK STRUCTURES.....	49
RECURRENT NEURAL NETWORKS	49
CONVOLUTIONAL NEURAL NETWORKS	52
4.2 THE HISTORY OF NEURAL MACHINE TRANSLATION	54
OUT-OF-VOCABULARY WORDS	57
4.3 NEURAL LEXICAL NORMALIZATION	58
CHAPTER 5 MORPHOSYNTACTIC NEURAL NETWORKS FOR GENERALIZED TEXT NORMALIZATION	61
5.1 ARCHITECTURE	61
PREPROCESSING.....	62
GENERATION OF NOISY WORD EMBEDDINGS	63
WORD-LEVEL CONTEXTUAL ANALYSIS	64
GENERATION OF OUTPUT NORMAL WORDS	64
5.2 CORPORA	65
TWITTER LEXICAL NORMALIZATION (W-NUT).....	66
OPTICAL CHARACTER RECOGNITION POST-PROCESSING (HOLMES)	66
AUTOMATIC SPEECH RECOGNITION POST-PROCESSING (LIBRISPEECH).....	67

5.3 EVALUATION	69
5.4 RESULTS	70
W-NUT.....	73
HOLMES.....	74
LIBRISPEECH.....	75
5.5 DISCUSSION	76
CHAPTER 6 ANCILLARY EXPERIMENTS	78
6.1 COX WINDOWING	78
6.2 TWEETCODE EXCLUSION	80
6.3 UNIFIED RECONSTRUCTION AND FLAGGING	83
6.4 UNSUPERVISED PRETRAINING	86
UNSUPERVISED LEARNING METHODS.....	87
UNSUPERVISED DATA FOR W-NUT	87
UNSUPERVISED DATA FOR HOLMES.....	88
UNSUPERVISED DATA FOR LIBRISPEECH	88
RESULTS	89
6.5 DOWNSTREAM ANALYSIS	93
6.6 DISCUSSION	94
CHAPTER 7 CONCLUSION	96
7.1 HYPOTHESES REVISITED	96
7.2 LIMITATIONS	99
7.3 FUTURE WORK	100
7.4 SUMMARY	101
REFERENCES	103
CHAPTER 8 APPENDICES	111

LIST OF TABLES

Table 1: Variations on the word “together” in Twitter	10
Table 2: Number of words corrected by an annotator in each dataset.....	36
Table 3: Results of the constrained task	44
Table 4: Model scores on validation and test data.....	44
Table 5: Flagger scores on validation and test data	44
Table 6: Analysis of errors. Percentages given are out of the total error count.....	45
Table 7: Examples of tokens that were mistakenly flagged "Do not normalize"	45
Table 8: CMU-Sphinx recognitions of speech (top row) as aligned to the ground truth text (bottom row)	68
Table 9: Evaluation results comparing character error rate between models	71
Table 10: Word-level metrics comparing models on W-NUT	71
Table 11: Word-level metrics comparing models on Holmes	72
Table 12: Word-level metrics comparing models on Librispeech.....	72
Table 13: Examples of one-to-many terms in W-NUT and the normalizations offered by each approach.....	74
Table 14: An example noisy tweet and the corrections offered by each approach.....	74
Table 15: An example sentence from the Holmes OCR post-processing corpus and the corrections offered by each approach	75
Table 16: Results on W-NUT with and without Cox windowing	79
Table 17: Results on Holmes with and without Cox windowing	80
Table 18: Effects of Tweetcode exclusion on results	82
Table 19: Separate output results versus unified on W-NUT	84
Table 20: Separate output results versus unified on Holmes.....	85
Table 21: Randomly Selected Tweets from Nist’s noisy and clean datasets.....	89
Table 22: Unsupervised pretraining results on W-NUT	93
Table 23: Unsupervised pretraining results on Holmes.....	93
Table 24: Unsupervised pretraining results on Librispeech	93
Table 25: Statistics on Stanford Part of Speech on normalized and raw Twitter data.....	94

LIST OF FIGURES

Figure 1: A simple example of word segmentation issues in ASR output	20
Figure 2: Two feed-forward neural networks of different depth	30
Figure 3: A flowchart detailing the process of normalizing a word	37
Figure 4: A diagram of the Reconstructor correcting "u" to "you"	38
Figure 5: The Reconstructor component validation scores by epoch	46
Figure 6: A diagram of a simple recurrent neural network.....	50
Figure 7: The gated recurrent unit	51
Figure 8: A narrow (top) and wide (bottom) convolution for a given headline	53
Figure 9: The recursive gated convolutional network	55
Figure 10: An automatic caption generator using NMT techniques.....	56
Figure 11: The annotation-based neural machine translation system.....	57
Figure 12: Flow of data through DeepNorm.....	62
Figure 13: The unified multi-softmax output layer.....	65
Figure 14: Text from the Holmes corpus after degradation.....	67
Figure 15: A comparison of the character error rate during training on W-NUT and Holmes	73
Figure 16: Character error rate on Librispeech during training.....	76
Figure 17: Effects of removing Cox windowing on training of W-NUT	79
Figure 18: Effects of removing Cox windowing on training of Holmes	80
Figure 19: Effects of Tweetcode exclusion on training.....	82
Figure 20: Flagger and Reconstructor training on W-NUT	85
Figure 21: Flagger and Reconstructor training on Holmes.....	86
Figure 22: Character error rate on supervised task during unsupervised pre-training.....	91
Figure 23: Performance during supervised training after unsupervised pre-training	92

CHAPTER 1

INTRODUCTION

As time passes, ever more data becomes available on the web, frequently in the form of text with little or no formal structure. This text comes from many sources, including newswire, blogs, speech transcripts, scans of printed documents, and social media and is known as unstructured text. Companies, organizations, and governments around the world understand the value of being able to extract useful information from large quantities of unstructured text and therefore research in text analytics is on the rise. The predominant techniques in text analytics, however, are primarily developed and evaluated on the Wall Street Journal and the Brown corpus. The text in these documents, referred to as *normal text*, follows formal rules of spelling and grammar that apply to only a small portion of the text available on the web (Han and Baldwin, 2011; Baldwin et al., 2015).

Meanwhile the text on the web takes a number of forms outside the realm of normal text. We refer to outside the boundaries of normal text as *noisy text*. Users write text in noisy forms and physical and verbal media are made noisy through imperfect recovery. We will elaborate on this process throughout this chapter. Noisy text poses significant challenges to techniques developed for normal text. In order to analyze it we must either revise the existing normal text techniques or develop a technique for converting noisy text into normal text.

There are different kinds of text normalization for fixing different kinds of noise, including correction of syntactic form, restoration of noisy words into their normal forms, restoration of lost punctuation and removal of inappropriate punctuation, capitalization correction, and phonetic conversion to support text to speech systems (Sproat et al., 2001; Goth, 2012; Mikheev, 2000; Shugrina, 2010).

In this work we will focus on the conversion of noisy words into their normal forms, lexical normalization. We chose this task due to the primacy of words in delivering meaning. Among punctuation, syntax, capitalization and words, the words themselves are the most significant source of meaning in text (Landauer and Foltz, 2012). This task is far from trivial, and we will describe some of the major challenges in detail in Section 1.2.

Many systems have been developed for lexical normalization, but each system has approached this general task from the perspective of a smaller, more specific task. Instead of lexical normalization in general, there have been approaches to correcting spelling errors, approaches to resolving issues in optical character recognition, approaches to resolving issues in automatic speech recognition, and approaches to normalizing Twitter text. To date, no approach has been developed and evaluated on combinations of these tasks to serve as a tool for lexical normalization in general. We propose to unite these disparate tasks under the single umbrella task of generalized lexical normalization. In doing so, we lay groundwork that can support research in the normalization of other, possibly unencountered types of noise. In this section we describe first the task of lexical normalization and the three tasks within lexical normalization that we intend to unite and approach in a general manner: unedited text, optical character recognition post-processing, and automatic speech recognition post-processing. Next we describe some of the challenges present in the task of lexical normalization. Finally we give our research question and our hypotheses, followed by a brief discussion regarding the competitive models we have chosen.

1.1 The Umbrella Task of Generalized Lexical Normalization

We define the Generalized Lexical Normalization umbrella task as follows:

- **Given:** Word or short phrase W in document D
- **Generate:** Word or short phrase W' such that
 - (a) W' adheres to a predefined *normal* form
 - (b) W' maintains as precisely as possible the meaning of W as it appears in D
- **Constraint:** One or both of W and W' must be a single word

We refer here to any individual token as a “word.” For example, take the sentence “the light was on.” Each of “the,” “light,” “was,” and “on” we refer to as a word, but the “.” is also a word for the purpose of this definition. The Twitter dataset W-NUT, which we will introduce in Chapter 3, defines punctuation as not to be corrected, but our OCR corpus, as we will describe in Chapter 5, does introduce minor punctuation errors that can be corrected by a

lexical normalization model. We use the term “word” instead of “token” because the vast majority of such tokens to be corrected will fall under the common definition of “word,” and minor punctuation errors are the exception.

“Word or short phrase” here refers to any word or grouping of consecutive words in the given document D . An example of a short phrase is “a lot” as a correction for “alot,” or “together” as a noisy version of “together.” Hyphenated words such as “problem-solving” are treated as one word in Baldwin et al.’s (2015) definition of lexical normalization for Twitter, so we will treat them as one word in our definition as well. The document D refers to a body of text of any size, although for our purposes we do not consider more than a sentence of text at a time.

In general, the normal form referred to in (a) is characterized by strict adherence to the formal rules of spelling and grammar,¹ although these rules change over time, such as with the addition of new words to the English lexicon. In addition, particular normalization tasks may demand exceptions to these rules and thus have their own variants of the normal form. The “normal” form is defined so as to best benefit common NLP systems, which have largely been trained on the Wall Street Journal and the Brown corpus.

An example of a task that demands a variant normal form is the normalization of Twitter text. Twitter’s interjectional abbreviations (“lol”), emoji (“:”) and codes (“#hashtag”) evoke specialized meanings unrepresented in the normal corpora, and, as such, in W-NUT, they are treated specially (Baldwin et al., 2015). Codes and emoji are left as-is in normal form, and interjectional abbreviations are expanded into the short phrases they abbreviate. In addition, the W-NUT task’s normal form permits graceful degradation. By this we mean that if an annotator cannot determine a correction for a noisy word, she should leave it as-is. (An annotator who finds “GNICNRFPO” in the noisy text should, for instance, include it without change in the normal version). Finally, for simplicity of evaluation the W-NUT normal form renders all output in lowercase (Baldwin et al., 2015).

¹ Only spelling for the purposes of lexical normalization

Rule (b) in general can easily be understood if we consider Lexical Normalization of W as a translation of W from a noisy form into a normal form. Just as a translation between languages should optimize to preserve as much meaning as possible, so should a conversion from noisy to normal form. Because noisy text and its normal form tend to be much more closely related to begin with than a text in one language and a text in another language, the preservation of meaning becomes stricter in the normalization task. For example, in a hypothetical tweet “my little boi iz growin up soooo fast <3 <3 <3,” offering the word “guy,” as the normalization of “boi,” while it could be an acceptable translation from another language, is not appropriate between noisy and normal English. The fact that the noisy word “boi” is morphologically more similar in form to “boy” than “guy” is relevant in normalization where it is not in translation, on the assumption that, in the absence of perfect understanding of meaning, attempts at preservation of morphology will help to preserve nuances that cannot be detected directly.

The degree to which meaning must be preserved in the process of normalization is simple in cases of technical noise such as ASR and OCR post-processing where the noising process is guaranteed to confer no additional meaning, but in normalizing the noise present in unedited text such as Twitter, meaning preservation remains an open question. Continuing with the Twitter example, when a user writes a tweet such as “my little boi iz growin up soooo fast <3 <3 <3,” she may be accidentally mistyping “growin” or deliberately representing dialect by not typing the final “g”, which she would not pronounce. Similarly, the elongation of “soooo” likely indicates emphasis. If we delve into pragmatics, the tweet could be intended to mock the style in which someone else has written her tweet. Comedian Sarah Silverman’s tweet “ya ur website suxx bro,” for example, is most meaningfully interpreted as intended to convey a caricatured persona (Eisenstein, 2013). Ideally, the normalization process would maintain all of this subtlety precisely, but this requires immense knowledge of specific and greater context as well as cultural understanding. Even with all of the necessary knowledge, one might be able to write a short essay exhaustively describing all of the nuances of this tweet in normal form, but that would not be the same as normalizing the tweet itself.

Fortunately, very few of the natural language processing tasks that we might want to support with normalization require this depth and precision of representation. If we are normalizing for the support of downstream tasks, which elements of meaning are most important to maintain will depend on the task in question. The change, “soooo,” → “so,” for example, loses its emphasis, which could be important in a graded sentiment analysis task, but would be less important in a named entity recognition task. An in-depth exploration of these issues is outside of the scope of this work. In fact, for our work the question has largely been answered already, since the only data sets that we use are technical noise datasets with unambiguous normal forms and our unedited user text dataset, the Twitter dataset, which has already been annotated according to the W-NUT Twitter normal form variant we describe above. In order to get a sense of how effective our model combined with this normal form variant is for assisting downstream NLP tasks, we explore the improvement on part of speech analysis between the original noisy data and the normalized form.

One reason lexical normalization is particularly important in support of standard natural language processing techniques is because in large part these techniques function at the word level, ignoring morphology. When a word is noisy, a word-level task trained on normal text may treat it as a different word or as if it has never seen it before. For example, in a word level model trained on noisy text, “teh,” a common typo for “the”, would be represented as unknown and lose all its meaning when even a simple character-level normalizer run as a preprocessor could correct it and allow it to be recognized with no loss.

The description of lexical normalization we have given above applies to many tasks, only a few of which refer to themselves as lexical normalization in the literature. This disunity of the general task has led to specialized approaches with manually engineered features that perform well in one specific task without generalizing to the general task. To resolve this, we propose to develop an architecture that performs tasks in the general domain of lexical normalization, selecting a small number of them to evaluate our model. Instead of manually developing a set of general features, however, we built a deep learning system for the general task that can, with a combination of annotated and unannotated training data, adapt itself to

each specific task. We refer to the task of performing these multiple specific lexical normalization tasks with the same model as *generalized lexical normalization*.

To treat this as a sequence labeling task wherein we label each noisy word with its normalization, we can modify our goal as follows. For each word W in document D , either return a word or phrase W' as above or return ϵ . Here ϵ represents the empty string and means that an input “word,” which in fact may only be a letter or part of a word, should be removed entirely in the normalized version. This permits us to associate a single normalization with each input word. The normalization of “l o v e” to “love” for example, can be represented as [“l”, “o”, “v”, “e”] \rightarrow [ϵ , ϵ , ϵ , “love.”]

As we will describe in more detail in Section 1.2, annotated data for lexical normalization is scarce. For that reason, we have selected three normalization tasks based on their broader impact as well as the availability of gold standard corpora for their evaluation. We introduce these tasks below. We describe the approaches to each of these different tasks in more detail in Chapter 2 and how we collect these corpora in Chapter 5.

Optical Character Recognition Post-processing

Optical Character Recognition (OCR) is the process of converting images of printed and typed text into digital text (Impedovo et al., 1991). Taghva and Stofsky (2001) describe the main sorts of optical character recognition (OCR) errors:

- **Unexpected Characters:** Errors can add numbers, punctuation, and other characters into words, making it more difficult to separate a word from a non-word token, such as a phone number.
- **No Conservation of Characters:** One-to-one substitution is not guaranteed (m can be read as iii).
- **Segmentation Challenges:** Words are often broken or combined, making tokenization non-trivial.

As of 2014, state-of-the-art OCR error rates range from 1-10% depending on the age and state of the document (Kata and Sagot, 2014). While this may be sufficient for indexing, effective digitization of documents for human consumption demands a further error reduction

by a factor of 10 to 100 (Kata and Sagot, 2014). Towards this end, OCR post-processing, using linguistic phenomena to correct the text generated by an OCR system, is well-represented in the literature (Bassil and Alwani, 2012; Kolak and Resnik, 2005; Lon-Mu et al., 1991; Llobet et al., 2010). In addition to being a current and important area of research, we choose the OCR post-processing task for evaluating our general lexical normalizer due to the relative ease of generating an aligned evaluation corpus of normal and noisy OCR data by rendering text, reducing the image quality, and recognizing it with OCR. We describe the corpus generation process in more detail in Section 5.1.

Automatic Speech Recognition Post-processing

Automatic speech recognition (ASR) is the process of converting digitized speech into text (Jurafsky and Martin, 2008; Manning and Schütze, 1999). This process is not perfect and even modern systems are prone to error. The current state of the art in speech recognition gets an error rate between about 5 and 10% (Novet, 2015). Here we present a list of some of the main sources of noise in ASR modified from (Bassil and Semaan, 2012):

- **Segmentation Challenges:** Words blending together in continuous speech (e.g., “buddai” instead of “but I”)
- **Disfluencies:** “um” and “uh” can be misrecognized as words, and awkward pauses can lead to a word being misrecognized as two separate words.
- **Variation in Pronunciation:** A mispronounced word or pronunciation variation can be misrecognized as a different word
- **Out of Vocabulary (OOV) errors:** Small, monolingual vocabularies make development feasible, and without severe restriction of domain, one cannot cover the vast array of possible named entities, so OOV errors are very frequent.

The process of analyzing text that has been generated by an ASR model to retroactively correct errors is known as post-correction or post-processing. In this dissertation we will refer to it as post-processing. A very similar task is decoding, which operates after the audio data has been translated into phonemes and converts the string of phonemes into words (Jurafsky and Martin, 2008). The motivation to use a post-processing approach instead is that it allows

the normalization of issues independent of the ASR system itself. For example, such an approach could normalize a collection of transcripts found to be noisy after their generation through an ASR system. ASR post-processing is well-represented in the literature (Twiefel et al., 2014; Huet et al., 2010; Ringger and Allen, 1996). ASR post-processing is also a task for which it is straightforward to obtain annotated data for evaluation given aligned text and speech. Such a corpus is freely available on the Internet through Librispeech, which we describe in more detail in Section 5.2, making ASR post-processing an ideal sample task for evaluating a general normalization system.

Twitter Normalization

Twitter normalization involves many of the same considerations as both OCR and ASR post-processing. Like OCR, an effective Twitter normalizer should be able to normalize words with dropped and substituted characters, even though due to coming from such a different source, they certainly will arise in different patterns. Similarly, the forms that words take on Twitter are often based on how they are pronounced, much like the errors in an ASR system, although the results of this pronunciation based distortion are distinctly different. Twitter’s sources of noise do not end here. From a qualitative study of Twitter text we formed a list of some of the most frequent types of Twitter noise:

- **Typos:** E.g., “troble”, “teh”
- **Phonetic Substitution:** E.g., “L8r”, “u r gr8”, “4EVA”, “trubl”
- **Dialectical Spellings:** E.g., “dem”, “dat”, “dose”, “da”
- **Expressive Capitalization:** E.g., “I am so MAD”
- **Vowel Elongation:** E.g., “Sooooooo Cooooool”, “Totallyyyyyy”
- **Expletive Infixation:** E.g., “Fan- f*****-tastic”
- **Phrasal Abbreviations:** E.g., “LOL”, “ROFL”, “LMFAO”, “SMH”
- **Emoji:** E.g., “:)”, “:(”, “O_o”

With over 500 million tweets posted per day, Twitter is a prolific source of user text invaluable to businesses and governments, but difficult to analyze because of its prodigious noise (Twitter, 2016). As such, the field of Twitter normalization has seen a particular surge

of interest in recent years (Han and Baldwin, 2011; Baldwin et al., 2015; Kaufmann, 2010). This interest led to the creation of a human-annotated Twitter normalization corpus that we use to evaluate our approach to general lexical normalization. We describe this corpus in more detail in Section 3.2.

1.2 Challenges

Despite years of research, lexical normalization remains an unsolved problem. As it exists now, the field is disjoint and treated as multiple separate fields, the definitions of what constitutes “normal form” remain ad-hoc, and it is difficult to find large corpora of training data. In this section we describe these three challenges in more detail.

Variety of Noise

The myriad forms and types of noise in text reflect the vast complexity of language itself. Some sources of noise are simply the result of machine errors, such as issues in automatic speech recognition (ASR), optical character recognition (OCR), or handwriting recognition. We will refer to these as *technical noise*. Other sources of noise arise from different manners of communication, such as text on social media, and text written by novice or non-native writers. This noise arises as part of the writing process and we therefore refer to it as *cognitive noise*, which carries its own set of challenges distinct from technical noise. It is important to note that cognitive noise can arise either unintentionally or intentionally, and just as often as an error, it is a valid alternative use of language. The normalization of cognitive and technical noise is further complicated by the fact that these sources of noise will overlap and can interact with each other in unpredictable ways.

As indicated above, cognitive and technical noise are two broad categories of a vast and expanding set of *noisy channels*. Here we use *noisy channel* to refer to a theoretical independent noising factor that plays a part in transforming a normal word into a noisy word (Brill and Moore, 2000). To explain noisy channels, we can take the general description of lexical normalization given in Section 1.1 and assume that the normal form of a word or phrase W is its original intended form. This original form passes through some combination of noisy

channels and produces a noisy word or phrase W . This assumption is the basis of the noisy channel model, which we will describe in more detail in Chapter 2.

As an example of just how many channels can be at work in Twitter alone, (Liu et al., 2012) found twelve different variations on the word “together,” in which one can quickly identify more than five different channels at work. These include a typographical noisy channel (together), a phonological substitution channel (2gether) that also includes particular dialectical pronunciations (togeda), a channel based on swapping characters of a similar appearance (t0gether), and a combination of these and other channels that are more difficult to categorize (togethaa). We show a list of these along with the number of times they appeared in the Edinburgh Twitter Corpus in Table 1 (Liu et al., 2012).

Table 1: Variations on the word “together” in Twitter

2gether (6326)	togetha (919)	tgthr (250)	togeda (20)
2getha (1266)	together (207)	t0gether (57)	togethaa (10)
2gthr (178)	together (94)	together (49)	2getter (10)

Please note that these different channels of noise all appear in just one normalization task. Other normalization tasks each have their own collection of noising effects. For some examples, OCR of “together” in a degraded document could miss the bars on a “t” and get “together,” whereas a person speaking quickly could mumble “together” to an ASR system and produce “dig Heather.” A novice writer could write “tugether” or a handwriting recognition system might misunderstand a too-short “h” and predict “togetner.” This proliferation of noisy channels has led the research community to focus on particular noisy tasks and as such their work requires extensive manual engineering to specialize to each new task within normalization. To date, a general approach to normalization has not been suggested. Making forward progress in this direction is the primary goal of our dissertation.

Absence of an Encompassing Standard Form

The normalization of cognitive noise is distinctly different from that of technical noise, and in many senses is much more difficult. The first challenge of cognitive normalization above technical normalization is that there is no concrete normal form to which to correct a text. Whereas technical noise comes from distortions that arise through imperfect conversion of language from one medium to another, with cognitive noise, it is not the medium but the language itself that is being converted. In effect, the “normalization” of text noised through cognitive channels is more akin to a translation task. In normalizing Twitter text, the preeminent example of mainly cognitively noisy text, the translation is from the Twitter dialect into another dialect of English; what we for the purposes of our work take as the normal form.

Approaches to defining a normal form for Twitter text are still evolving. In their 2011 normalization corpus, Han & Baldwin (2011) leave acronyms like “LOL” as they are to make their normalization task simpler, while in 2015 they correct “LOL” to “laughing out loud.” They do this despite the fact that neither of these approaches returns a sentence that would look like it belonged in a column of the Wall Street Journal or even in the fiction section of the Brown Corpus (Baldwin et al., 2015). Given these hurdles, targeting the Wall Street Journal and Brown Corpus as the form for the normalization of Twitter and other social media text is likely to remain an ideal rather than a hard and fast standard.

Because the normal forms proposed remain ad-hoc and are theoretically based on increasing the effectiveness of conventional NLP techniques, there is room in the literature for an examination of the degree to which automatic normalization based on these standards can improve the performance of NLP techniques on noisy text.

Scarcity of Labeled Training Data

It is possible to generate supervised training data for some technical noise with no human component. This is accomplished by taking a digital document in standard form, the gold standard, and converting it into another form and back again, at which point it will include all the noise of the conversion process. For example, one can render a digital text document as an image, reduce the image quality to simulate document degradation, and then attempt to

recognize it using OCR (Zi and Doermann, 2004). The analogous technique for ASR works less well, as errors are unlikely to be representative of the idiosyncrasies of human speech.

Cognitive noise, because it by definition arises from the writing process itself, cannot be efficiently generated through this method. Instead, development of an annotated corpus of cognitive noise must work in the opposite direction. Starting with a noisy text from a source such as Twitter, the data must then be manually normalized by a team of humans. Because this process is laborious and expensive, there is a shortage of annotated corpora in cognitive noise datasets such as Twitter. Furthermore, a new, previously unconsidered source of noisy data will be expensive to annotate manually. Therefore it is worthwhile to explore how unsupervised data for a given normalization task and annotated data on different normalization tasks can be leveraged to support the given normalization task with limited annotated data available.

1.3 Research Question and Hypotheses

Research Question:

To what degree can the various tasks involving correcting noisy words into a normal form be combined under the umbrella task of lexical normalization, and how well can a single deep learning architecture address them as different instantiations of this generalized task?

For the purposes of this work, the “architecture” of a model remains the same when sources of training data and hyperparameters are changed. For example, a single architecture remains the same even if the number of nodes or layers is changed or new training data is added, but it is different if new hand-engineered features are added or a different machine learning technique is used. Although multiple architectures were considered in producing this work, for our main evaluation we select one architecture to run on all three tasks.

The “umbrella task of generalized lexical normalization” refers to the collection of tasks that fit the definition given in Section 1.1. The modalities we investigate include two sources of technical noise (ASR and OCR output) and one source of cognitive noise (unedited text from Twitter). The specific tasks we have selected to represent each of these modalities are, respectively, automatic speech recognition post-processing on public domain readings of various classic novels, optical character recognition post-processing on *The Adventures of*

Sherlock Holmes, and correction of tweets into the normal form variant for the W-NUT competition described in Section 1.1.

The research approach involves collecting corpora of noisy text and their normalized counterparts, and developing a normalization system by extending neural machine translation techniques via other deep neural network techniques to make it applicable to the general lexical normalization task. We have three corpora on which we evaluate our approach and two models against which we compare. Thus, our primary hypotheses are as follows:

H1_a. A single end-to-end architecture based on neural techniques can perform a supervised lexical normalization task on unedited text competitively with a state of the art skip-bigram model.

H1_b. A single end-to-end architecture based on neural techniques can perform a supervised lexical normalization task on unedited text competitively with an off-the-shelf optical character recognition post-processing model.

H2_a. The architecture from Hypothesis 1 will perform competitively with the skip-bigram model on the supervised task of normalizing optical character recognition output.

H2_b. The architecture from Hypothesis 1 will perform competitively with the optical character recognition post-processing model on the supervised task of normalizing optical character recognition output.

H3_a. The architecture from Hypothesis 1 will perform competitively with the skip-bigram model on the supervised task of normalizing automatic speech recognition output.

H3_b. The architecture from Hypothesis 1 will perform competitively with the optical character recognition post-processing model on the supervised task of normalizing automatic speech recognition output.

As secondary work, we also explore the potential for learning from unannotated data and from data annotated for different normalization tasks. Our secondary hypotheses are thus as follows:

H4: There exists a combination of annotated unedited text, annotated text from other tasks, and unannotated text such that the architecture from Hypothesis 1, trained on this data and evaluated on unedited text, will achieve significantly higher performance than the same model trained using only the evaluation task’s annotated text.

H5: There exists a combination of annotated unedited text, annotated text from other tasks, and unannotated text such that the architecture from Hypothesis 1, trained on this data and evaluated on optical character recognition text, will achieve significantly higher performance than the same model trained using only the evaluation task’s annotated text.

H6: There exists a combination of annotated unedited text, annotated text from other tasks, and unannotated text such that the architecture from Hypothesis 1, trained on this data and evaluated on automatic speech recognition text, will achieve significantly higher performance than the same model trained using only the evaluation task’s annotated text.

Tertiary work is to explore the effects of the combination of the normalization standard and the normalization model on downstream tasks. We take our normalization standard from W-NUT, as described in Section 1.1, and use part of speech tagging as our downstream task.

On two out of three tasks we are testing our approach on brand new data. This is necessarily data on which no other model has been run. We compare to a skip-bigram model for Twitter normalization and CLSTM, a model for OCR error correction (Jin, 2015; Azawi et al., 2014). The former approach is the current state-of-the-art on the W-NUT Twitter task. The latter is a published deep learning approach to OCR error correction.

Our approach is specifically designed to be generalizable and portable in that it uses no hand-engineered features or off-the-shelf tools. Thus, although it may not outperform the skip-bigram model on Twitter, we expect it will perform competitively with the skip-bigram model on all three tasks. We represent this expectation in our first three hypotheses in Section 1.3. In Chapter 6, we compare different permutations of our architectural framework to isolate the effects that different modules have on results.

1.4 Contributions

This dissertation work has added to the literature in natural language processing and deep learning. Novel contributions of this work include the following:

- C1. A new task: Generalized lexical normalization. We propose to treat the collection of lexical normalization tasks, ASR post-processing, OCR post-processing, Twitter lexical normalization, and other such tasks hitherto approached separately as all domains within a larger task of lexical normalization in general. Before this work there was little cross-referencing in papers describing approaches to these tasks.
- C2. A deep feed-forward neural network that performs competitively on an international level in lexical normalization for Twitter. This is the first feed-forward neural network to be applied to lexical normalization (Leeman-Munk et al., 2015).
- C3. A reconstruction-based word prediction output for lexical normalization. The notion of constructing words as sequences of letters in output rather than selecting from a large vocabulary has not previously been proposed in the literature (Leeman-Munk et al., 2015).
- C4. Extensive analysis of the relevant preferred parameters and existing issues surrounding normalization using a deep learning framework on a small dataset (Leeman-Munk et al., 2015).
- C5. A versatile normalization system that works simultaneously at the character level and the word level. This approach does not require feature engineering for application to different tasks within general lexical normalization and can effectively normalize each of Twitter and OCR noisy text given a moderate quantity of training data in each realm.

C6. A noisy ASR text corpus generated from LibriSpeech, with the original text of the novels themselves serving as the gold standard.

C7. A noisy OCR text corpus generated by rendering classic novel text in a low-quality image and extracting the text again using OCR. The original text serves as the gold standard.

1.5 Organization

The remainder of this document is structured as follows. Chapter 2 describes the current state of the field of lexical normalization and makes a case for bringing the field together and developing a system for generalized lexical normalization. Chapter 3 introduces neural networks and describes work on using them for Twitter Normalization. Chapter 4 describes advanced deep learning techniques and the recent field of neural machine translation on which we partly base our normalization system. Chapter 5 presents DeepNorm, a morphosyntactic network for generalized lexical normalization. We describe the corpora on which it was run and present and discuss results. Chapter 6 describes ancillary experiments that explore how different aspects of our architecture affect task performance. It is also in this chapter that we present our study on semi supervised and unsupervised learning. Chapter 7 summarizes our contributions, presents conclusions and suggests directions for future work.

CHAPTER 2

RELATED WORK: LEXICAL NORMALIZATION

Although each of the tasks described in Section 1.1 has a rich literature of normalization behind it, these literatures do not overlap. To date there is no method that attempts lexical normalization on noisy text from more than one source. This chapter describes some techniques for spelling correction, an early and simple example of lexical normalization, then moves onto each of the three distinct tasks that we propose to use to evaluate our general lexical normalizer. To explain our choice of organization, we will begin by describing the noisy channel model, which provides a useful duality of morphology and context for understanding lexical normalization.

The noisy channel model, from which the concept of noise as separating into channels arises, operates on the assumption that a given observed noisy text arises from a combination of an underlying normal text and one or more noisy channels. Given this assumption and the observed noisy word W' , one can estimate the probability that the original word was W using Bayes' rule (Kernighan et al., 1990; Box and Tiao, 1992; Shannon, 1948):

$$P(W|W') = \frac{P(W'|W)P(W)}{P(W')}$$

Given that our goal is to select the most likely W , and $P(W')$ is the same for all W this value can be safely ignored. The expression for finding the most likely word using the noisy channel model is thus as follows:

$$\operatorname{argmax}_W P(W'|W) P(W)$$

With this assumption, what the noisy channel model needs to estimate is $P(W'|W)$ and $P(W)$. $P(W)$ can come from language models based on the surrounding words and how common the word A is in a large corpus. $P(W'|W)$ on the other hand, comes from the morphological characteristics of W' . In Section 2.1 we introduce the morphological models that have been used in various normalization tasks and explain the adjustments that have been made to the feature sets to specialize to each task and in Section 2.2 we discuss different approaches that have been taken to calculate $P(W)$ using the surrounding words, and describe how and why these techniques are largely similar between tasks. Section 2.3 describes what

to our knowledge is the first shared task on lexical normalization, and Section 2.4 describes machine translation techniques that have been used for lexical normalization. Section 2.5 illustrates the limited exploration thus far into general models of lexical normalization and draws upon the previous sections to explain why a general model now is both challenging and feasible. Section 2.6 explicitly lays out the novelty of this work.

2.1 Noise Models

The simplest approach to calculating $P(W'|W)$, which does not require explicitly selecting a noisy channel, is to build what is called a confusion set directly from an annotated corpus of noisy words and their normalizations. To build a confusion set, one counts the number of times in an annotated corpus that each word W' in the vocabulary is confused for W . This will give a rough estimate of $P(W'|W)$, but cannot generalize to any confusion that did not explicitly appear in the corpus. Calculating $P(W'|W)$ in a more robust manner is where the noisy channels themselves come into play.

Among the earliest examples of the noisy channel model targeted at the typographical and cognitive spelling error channel uses four separate confusion matrices to rank possible corrections for a given spelling error. The model supports one-character errors including additions, deletions, substitutions, and transpositions ('xy' \rightarrow 'yx'). Each error type has a confusion matrix. We give the details of the four matrices below. Here we use X and Y to represent arbitrary characters:

- Addition of X after Y
- Deletion of Y after X
- Substitution of X for Y
- Transposition of X for Y

This approach is more versatile for calculating $P(W'|W)$ than the no-channel approach because regardless of whether a given misspelling has appeared in a training corpus, its probability can be estimated if the particular typo that created it did appear (Kernighan et al., 1990).

Later work improved on this model. Brill and Moore (2000) present a noisy channel model that separates each word into partitions to allow a more generalized approach to spelling correction. Brill and Moore's approach to modeling spelling errors holds two advantages over Kernighan et al.'s model. Firstly, it allows multiplication of multiple edit probabilities together to estimate the probability of errors that are more than one edit away from the original word. They give the example "actual" \rightarrow "akgsual" which could be calculated based on the product of the probabilities "c" \rightarrow "k" "e" \rightarrow "g" and "t" \rightarrow "s." Second, they introduce character-level context to help inform likely errors. For example, at the letter "c" in the prior example, the probability of "c" becoming "k" is represented not only as "c" \rightarrow "k" but as "ac" \rightarrow "ak", "c" \rightarrow "kg", "ac" \rightarrow "akg", and "ct" \rightarrow "kgs." They also include positional information within the word, as one would naturally expect "confident" \rightarrow "confidant" to be more likely than "antler" \rightarrow "entler." Brill and Moore use a dynamic programming algorithm, the Wagner-Fischer algorithm, to select the most likely from multiple possible combinations of edits (Wagner and Fischer, 1974).

Toutanova & Moore (2002) expand upon Brill and Moore's approach by explicitly modelling two different noisy channels, one for typographical errors, and one for cognitive errors based on pronunciation. Using a letter-to-phone model, they predict phonetic sequences for the correct spelling of a word and the erroneous spelling and use largely the same technique as above for estimating the likelihood of one phoneme sequence becoming another. They combine their two channel models using a coefficient to allow tuning of which channel more strongly affects the prediction, but do not describe how this coefficient is either learned or selected.

Since OCR errors tend to be graphemic in nature, the noisy channel model has been used effectively in OCR with little change from the character edit models described above. Kata & Sagot (2014) use the simple single-character noisy channel model as part of a pipeline for OCR post-processing including named entity recognition and part of speech tagging. Tong and Evans (1996) use a noisy channel closer to Brill and Moore's. They use a dynamic programming method to support the possibility of multiple-character errors, but do not account for character context as in Brill and Moore. Kolak and Resnik (2005) compare Tong and

Evans's multi-character model and Brill and Moore's character-context model in their OCR work on the Nigerian language of Igbo.

Like OCR, ASR has also taken advantage of some noisy channel approaches, but because in ASR there are no characters to be individually misrecognized, these approaches generally eschew the graphemic approach in favor of phonetics. For example, Hacker & Noth (2014) use a phonetic model comparable to Toutanova and Moore's. Ringger & Allen (1996) proposed a model for ASR error correction that used two channels. The first was the basic word-for-word confusion set that simply counts the number of times a given word is confused for another given word. The second channel was a fertility channel for dealing with the segmentation challenges of continuous speech that estimated the likelihood that a word would be mistakenly split into two words or combined into one word. Figure 1 shows an example of word segmentation issues in ASR output (Ringger and Allen, 1996).

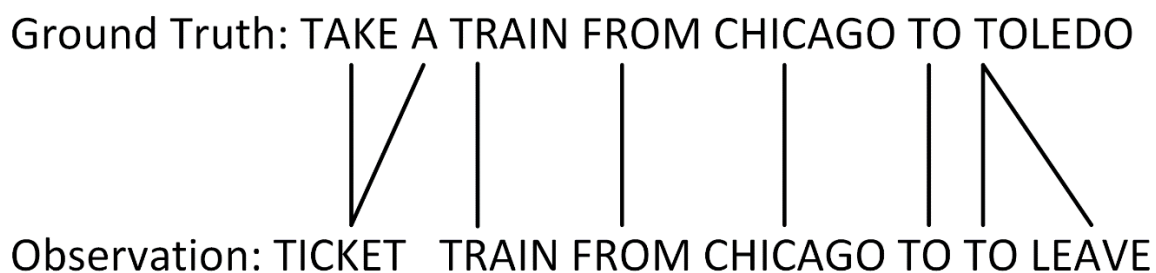


Figure 1: A simple example of word segmentation issues in ASR output

Twitter required some enhancements over prior models as well, particularly because it features more frequent, dense, and varied use of alternative spellings. Choudhury et al. (2007) propose a supervised noisy channel based on HMMs for normalizing words with arbitrary combinations of phonetic and character noise. They explicitly separate the cognitive channel and the typographical channel with the cognitive channel serving as HMM transmissions and the typographical represented in emissions. They train their model on 20,000 annotated SMS text messages. Since text messaging is, like Twitter, an un-refereed, limited-character medium, later works have referred to this text message corpus as an analog for Twitter normalization (Choudhury et al., 2007; Chrupala, 2014; Han, 2014).

Further work on the Twitter lexical normalization task comes from Cook & Stevenson (2009), who proposed an error generation model for a few channels of noise, some very specific, including phonetic spelling errors and clipping, which is the removal of letters at the end of a word (“walkin” → “walking”). Xue, Yin, & Davison (2011) add support for common abbreviations that expand to multiple words such as (“tlyl” → “talk to you later”). Instead of learning from data, they use a pre-existing list of common abbreviations².

Because it is cognitive and therefore exceptionally labor-intensive to annotate, Twitter lexical normalization has spurred research in unsupervised learning methods. Han & Baldwin (2011) use a 1.5 GB corpus created by filtering out tweets that include non-standard words to train their model. They normalize words based on contextual and morphophonetic similarity to the words in a dictionary. Han, Cook, & Baldwin (2012) take a similar approach, but use the technique to construct a lightweight mapping dictionary between words and their “lexical variants.” Each of these approaches focuses only on phonetic and morphological correction, excluding, for example, abbreviations such as “lol”. Gouws et al. (2011) develop a technique to deal with this weakness based on “exception dictionaries” mined from distributionally similar Twitter words, meaning the words tend to occur in the same context within a large body of text. Gouws et al. (2011) thin the exception dictionary, which at this point is full of matches like “I” → “you” and “and” → “but,” by removing matches that are also distributionally similar in normal text. Finally, using longest common subsequence and other string similarity measures, they further winnow the set. They demonstrate that the addition of this exception dictionary significantly improves the performance of a typical normalization system. Because it does not support many-to-one translations, however, it still cannot normalize “lol” and similar phrasal acronyms.

2.2 Language Models

As we mentioned above, $P(W'|W)$ is calculated from the morphology of W combined with a model of the noisy channel itself. Similarly, the words in a sentence provide contextual cues

² http://www.webopedia.com/quick_ref/textmessageabbreviations.asp

that support calculation of $P(W)$. Although context is helpful for word correction in many circumstances, a few examples of cases in which a context language model is especially important include real-word spelling errors, which appear in the dictionary but are not the intended word and cases in which one word has erroneously become split into two. When the language model is based on the target, normal language, it does not depend on the particular source of noise, and thus many systems use the same techniques for their language models. In this section we provide an overview of language models that have been used in normalization.

In their spelling-correction system, Toutanova & Moore (2002) assume a uniform language model in which $P(W)$ is assumed to be the same for all words and is thus not calculated. Their claim is not that the language model is not useful, simply that it was not the focus of their work. Kernighan et al. (1990) based their language model on the counted occurrences of a given word in a large corpus. Although it does not use any context, this approach is still useful in many cases because a large number of words in the English language are very uncommon. For example, the misspelling “confident” → “confidant” can be corrected with a reasonable degree of accuracy based on the knowledge that the former word is much more likely to be used, all other variables equal. Nevertheless, this approach will not fare well in the sentence “He was her best friend and confidant in times of trouble,” which will benefit from context. Brill & Moore's (2000) trigram language model would be likely to allow them to successfully avoid miscorrecting “confidant” in the above context. The trigram model uses the two words prior to a given word as context to inform a language model. In this example, “friend and” would be the context. It is called the trigram model because including the word in question, there are three words being considered at a time. Any number of words may be considered at a time. When not specifying the number of words considered, this is known as an n-gram model (Brown et al., 1992).

Context on a per-word basis is effective for isolated spelling errors in the presence of largely correct text. However, imagine if in our earlier example, instead of “friend and confidant” the text were “frnd n confidant.” An n-gram model based on the target normal language would not be able to estimate a probability that “confidant” is the correct form in the given context. We need to correct the words around a word before we can correct a word, but

often context is bidirectional, and the words that are needed to correct a given word themselves need the word that we are trying to correct!

To deal with this issue, we must consider all words in a given sentence at once and select the corrections that maximize the whole sentence. Goth (2012) describes a “language-noise” model based on a finite state transducer that traverses the possible corrections of each word in a given sentence and finds the path of minimum cost.

Language models are especially deeply investigated in ASR post-processing. Because the output of a typical speech recognition system is already converted into real words, spelling errors are almost all if not all real-word errors, making a language model much more important for detection and correction. None of the automatic speech recognition systems we reviewed for this dissertation operate without a context-dependent language model. Ringger & Allen (1996) use a bigram language model trained on a corpus of speech transcripts. Bassil & Semaan (2012) use the Microsoft N-Gram corpus, which has the advantage of size. Their system takes the unigram corpus as a vocabulary and uses the 5-gram corpus as context to evaluate $P(W)$ given the two words on either side of the observed word.

Rather than N-grams, Sarma & Palmer (2004) use information retrieval techniques on a corpus from 1.6 million words of existing ASR transcript data. By identifying how often two given words appear together versus separately they can evaluate how related those terms are. Using this technique one can take advantage of the huge corpora of Internet search engines for language modeling (Byambakhishig et al., 2014).

Tong & Evans (1996) and Kata & Sagot (2014) both use a bigram language model to support OCR post-processing. Bassil & Alwani (2012) base their model on Google’s *did you mean* spell correction feature, which, while effective, is fragile since it depends on Google to keep that feature available.

Iakovidis, Keramidas, & Maroulis (2008) use a long short-term memory (LSTM) network for supervised OCR post-processing (Hochreiter and Schmidhuber, 1997). They use a bi-directional LSTM that allows their model to use the entire sentence as context in both directions. They find that their LSTM achieves a lower error rate than both the weighted finite

state transducer and a rule based model on English and Urdu script, with the improvement being much more dramatic on the more difficult Urdu.

Twitter’s language models are very similar to those of basic spelling correction. Like OCR, some work uses no language model (Choudhury et al., 2007) and other work uses n-gram models (Pennell and Liu, 2011b; Xue et al., 2011). Chrupala (2014) argues that a language model based on target text cannot work for Twitter normalization because beyond being normal it does not match the way language is used in Twitter. For this reason, he uses a recurrent neural network trained on a large quantity of unannotated Twitter data and uses it to inform a conditional random field-based noise-language model for Twitter normalization.

2.3 The W-NUT Lexical Normalization for English Tweets Challenge

The W-NUT Lexical Normalization for English Tweets Challenge, described in detail in Section 3.2, presented Lexical Normalization as a supervised task, providing annotated training data that included matches between phrasal acronyms such as “lol” and their expanded counterparts. The two top-performing approaches on the constrained task that required that only the training data and no external data be used were explicit augmentations of a term mapping technique. As an exploration, we recreated this term-mapping baseline that used a mapping approach to match each noisy term to its most common normalized counterpart in the training data. If there was no normalization, we left the word as-is. This is effectively the word-to-word noise model described at the beginning of Section 2.1 with no language model-based prior. This extremely simple baseline achieved 80.4% F-score on the test data, outperforming half of the participants in the task. Because it is much more conservative than other models, normalizing only if there is an explicit mapping in the data, this baseline model achieves 93.3% precision, higher than any model submitted in either subtask. The models that augmented it used more sophisticated techniques to correct in the case of multiple mappings or no mappings.

The winning approach disambiguated between candidates provided by the term-mapping baseline using a random forest model with string similarity statistics and a part of speech (POS) tagger-based disambiguation technique. The POS tagger was applied to the tweet with each candidate normalization, and the confidence for the tag was used as a feature. An analysis of the data showed that the correct word would have a higher confidence score 90% of the time

(Jin, 2015). The second place approach used an LSTM based on character edit operations. It uses the LSTM for terms with more than one mapping or with no mapping (Min and Mott, 2014).

2.4 Machine Translation for Lexical normalization

If we consider English Twitter text as a particular dialect of English and normal text as another dialect, it should not come as a surprise that machine translation is frequently presented in the literature as an analogy for lexical normalization in Twitter. It is interesting to note that this is not the case for ASR and OCR. In Twitter normalization alone our literature survey found researchers taking advantage of machine translation techniques.

Aw, Zhang, Xiao, & Su (2006) use a phrase-based model inspired by statistical machine translation for normalization of SMS text. The key insight of this model is one that we have already discussed — that many phrasal acronyms in Twitter do not match cleanly to just one word and thus must be mappable to multiple words. A later approach uses an off-the-shelf machine translation system at the character level to match abbreviations to their full forms (Pennell and Liu, 2011a). Another approach uses the same statistical machine translation technique, and as an indication of the challenge that text annotation presents, also describes a simple interface allowing users to provide annotations to support the system (Schlippe et al., 2010).

2.5 General Models of Lexical Normalization

Based on our literature review, no model attempts general word correction on Twitter text, OCR output, and ASR output with a single architecture. Only a few models have been evaluated on multiple sources of noise at all. Mikheev (2000) describes his work in sentence boundary disambiguation, disambiguation of capitalized words, and identification of abbreviations as “domain independent”, but the various domains all come from the standard sources (WSJ and Brown corpora).

The most complete analysis of general lexical normalization comes from Sproat et al. (2001). For the purposes of supporting both ASR systems and text-to-speech systems, they give a definition of non-standard words broader than the one we use in this work. We correct

words into canonical forms that would appear in newswire text whereas their task is to convert all words into forms that directly represent their pronunciations (e.g., 100 → one hundred). They present an extensive taxonomy of non-standard words collected from newswire text, newsgroups for recipes and hardware, and real estate classified ads, included in Appendix 1. As their work predates Twitter, Twitter text is not covered in the taxonomy.

The similarity of the word correction tasks described in this section suggests an opportunity for a general approach that is not addressed in the literature, while the differences indicate that the development of a general technique that does not rely on task-specific channel or feature engineering will be a significant technical contribution. Deep neural networks have the ability to learn features from data, and are thus capable of approaching in a general manner tasks that once required specialized feature engineering. In the next section we describe deep neural networks' application to text, focusing especially on a field rapidly expanding over the past five years that serves as an informative parallel to lexical normalization, neural machine translation.

2.6 Novelty of this Work

Our task is novel because it will be the first requiring a single architecture to address three distinct tasks under the umbrella of lexical normalization.

The novelty of the architecture itself consists of four parts:

- a) It is a neural network that works at both the character and word level simultaneously.
- b) It does not rely on hand-engineered features.
- c) It does not rely on off-the-shelf tools.
- d) It is specifically designed to be applicable to multiple noise from multiple sources.

(a) Operating simultaneously at the character and word levels is important because previous approaches have shown strengths of each of word and character-level models that are not represented in the other (Chrupała, 2013; Mikolov et al., 2012). While this is not the first neural model to work at both the word and character level simultaneously, it is the first such to be applied to lexical normalization.

(b) The absence of hand-engineered features holds promise that our approach will not only work on the three datasets that we will use to evaluate it but on other datasets as well. Although we will be constructing our model to optimize on these datasets, we do so without any hand-engineered features. We instead allow the deep learning to automatically decide select relevant features from the data. We expect this to improve generalizability to datasets to which we do not have access.

It is important to note that we are not constructing a fully general model for this work. We will choose training datasets, supervised and unsupervised, to specialize to each task. For instance, in developing a normalizer for Twitter in our preliminary work we separated out the “tweetcode” (e.g. “#hashtag,” “@atmention”, and “http://url.com”) from the rest of the data and trained without it, relying on the flagger to learn that the rules of the task specified that these constructions were not to be normalized. To show the effects of including or excluding tweetcode when training a Twitter normalizer, we include an experiment in Chapter 6.

(c) Off-the-shelf tools are often helpful and easy to use, but their use can also be complicated by license restrictions or dependencies on still other tools. As operating systems progress and change, old off-the-shelf tools may cease to function, with the risk becoming more pronounced with each additional dependency an approach has. Thus, excluding off-the-shelf tools improves the portability of our approach. For the purposes of prototyping we will be using the programming language Python, a high-performance computational library Theano, a GPU programming library CUDA, and various other simple Python libraries, but we do not count these as off-the-shelf tools as we describe them here because they do not provide any sophisticated NLP functions and only serve to help us implement and test our architecture.

(d) To our knowledge this will be the first lexical normalization architecture explicitly designed for and evaluated on noisy text from multiple modalities.

With regard to deep learning’s prior applications to lexical normalization, although it has been applied to both optical character recognition post-processing and unedited text normalization in the form of Twitter, these approaches do not have the fully deep, generalized approach we will use here. Chrupala (2014) used neural networks as only part of his solution.

He used recurrent neural networks to provide additional features to an n-gram-based conditional random field (CRF) for learning Levenshtein-style edits for Twitter text whereas we will rely mainly on neural networks. Min & Mott (2015) use a neural network with a combination of information directly from the data and hand-engineered features collected from a Twitter-specific off-the-shelf tool. Our approach will not rely on hand-engineered features or off-the-shelf tools. In OCR post-processing, Bassil & Alwani (2012) use an LSTM approach that learns correction edits (insert, remove, transpose, replace) at the character level much like Chrupala and Min & Mott. Rather than restricting itself to the character level, our model works at both the character and word level as we will describe in more detail in Chapter 5.

CHAPTER 3

FEED-FORWARD NEURAL NETWORKS FOR THE NORMALIZATION OF TWITTER DATA WITHOUT CONTEXT

To explore the potential of neural network-based lexical normalization, we have developed an approach to transform noisy user-generated text into a canonical form with a feed-forward neural network augmented with a projection layer (Kalchbrenner et al., 2014; Collobert et al., 2011; Vincent et al., 2008). The model performs a character-level analysis on each word of the input and attempts to reconstruct the word in its correct form. We tested our approach on Twitter text and submitted it to the W-NUT Lexical Normalization for English Tweets Challenge. The absence of hand-engineered features and the avoidance of direct and indirect external data make this model unique among the three top-performing models in the constrained task. It is also unique in that it does not take advantage of context. Each morphologically unique input will be normalized to the same word regardless of its surrounding words.

We argue that *deep learning*, the general term for neural networks with multiple layers, is the current best approach to the umbrella task of lexical normalization. Deep learning is effective for generalization because it has achieved several competitive and state-of-the-art results in text analytics without the use of hand-engineered features (Kalchbrenner et al., 2014; Kim et al., 2015). While there are still approaches based on years of careful feature engineering that outperform deep learning in some text analytics tasks, the trend is for deep learning to close that gap, and it has done so remarkably quickly on many disparate tasks at once. We consider this to be sufficient evidence that deep learning is worth exploring for the umbrella task of lexical normalization.

In Section 3.1 we introduce deep learning for text analytics and the basic techniques involved in our approach. In Section 3.2 we describe the W-NUT Lexical Normalization for English Tweets Challenge. In Section 3.3 we describe each component of our model. In Section 3.4 we describe the specific instantiation of our model, and in Section 3.5 we present

and discuss results. In Section 3.6 we address some potential questions regarding the absence of feature engineering in more detail before we conclude with a summary in Section 3.7.

3.1 A Brief Overview of Deep Neural Networks for Text Analytics

Deep learning refers to the process of recognizing complex patterns in data using a series of multiple learned transformations. Each of these transformations serves as a feature collection algorithm for the next transformation, and collectively they permit a network to form deep representations of information that have a nuanced, non-linear relationship with the input data. In the simplest example, the feed-forward neural network (FFNN), these transformations are each represented by what is referred to as a “layer,” with more layers indicating more depth, but in more complex neural network architectures, multiple transformations may occur through recursive application of the same layer. See Figure 2 for an example of feed-forward networks of different depth. Input starts at the red nodes (circles) and forward-propagates, generating output at the yellow nodes. Blue circles indicate the nodes in hidden layers (LeCun et al., 2015). In this section we describe some of the basics of deep learning as applied to text analytics, leaving more advanced models for Section 4.1.

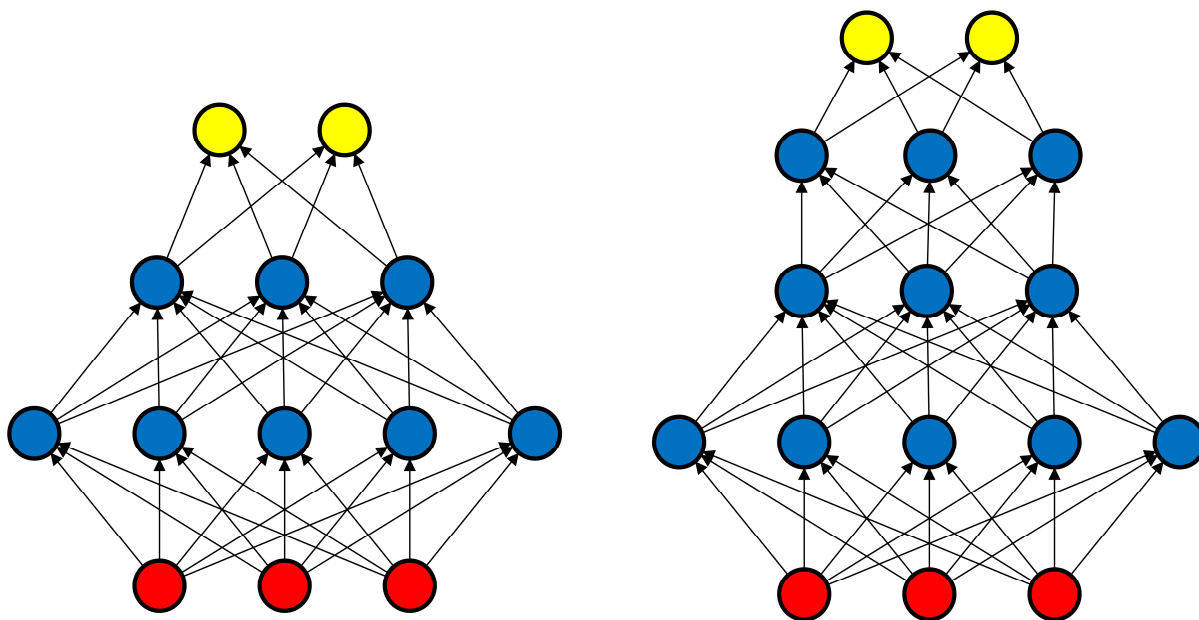


Figure 2: Two feed-forward neural networks of different depth

Deep Learning Fundamentals

Training deep neural networks relies on an algorithm known as backpropagation. Backpropagation determines the gradient of change at each weight in a network that will decrease the error between the prediction and the gold standard. With these gradients a neural network's weights can be initialized randomly and learned by one of any number of gradient descent algorithms (LeCun et al., 2015).

The simplest form of a deep learning algorithm is the deep FFNN. The FFNN takes a vector of numbers as input. This vector is known as a *layer* and each value within it is a *neuron*. The network multiplies the input layer by a matrix of weights to return another vector. This new vector is then transformed by a non-linear activation function. A number of functions can serve as the non-linearity, including the sigmoid and the hyperbolic tangent.

The transformed vector is referred to as a *hidden* layer because its values are never directly observed in the normal functioning of the model. A deep feed-forward neural network can contain any number of hidden layers, each going through the same process, multiplying by a matrix of weights and transforming via a non-linearity. Hidden layers may also be of any size. Multiple applications of learnable weight matrices and non-linear transformations together allow a deep neural network to represent complex relationships between input and output (Bengio, 2009).

Although one can train a neural network directly from randomly initialized values, to get the best performance, pre-training is important, especially in a deep neural network. Pre-training is the process of learning initial weights prior to the training of the final model, and helps a deep neural network in a number of ways, primarily helping it to avoid inferior local optima and to generalize to unseen data. To pre-train a neural network, the weights are first trained in another network. The pre-training network can be trained for the same task on the same data, but usually is trained on a different but similar task. For instance, a Twitter normalization model might benefit from pre-training on an OCR normalization task. The weights from the pre-training network are used to initialize their corresponding weights in the main network, and any unused weights are discarded. Once the weights in the main network

have been initialized, they continue to be learned as if they had been initialized randomly (Bengio, 2009).

There are many techniques for pre-training. Here we will discuss two techniques that are particularly relevant to generalized lexical normalization: Transfer pre-training and layerwise pre-training. Transfer pre-training refers to pre-training a network intended for one task, the target task, using a different task, the *base* task. The motivation to pre-train on a different task than one intends to solve is that the base task may have more data available. An unsupervised base task, for example, is very likely to have much more data available to train it than its supervised target task. For example, a network for part of speech tagging could pre-train on unannotated data by learning to predict the next word in a sentence (Yosinski et al., 2014).

Layerwise pre-training is so named because it trains the network layer by layer. There are two ways to do this, supervised and unsupervised. Supervised layerwise pre-training is the simpler of the two approaches because the task remains the same. In supervised layerwise pre-training, the model is trained first with some minimum number of layers, then with one additional layer at a time until the full model is constructed. In order to do this, one new layer is required for each new pre-training model. For example, if you have a 4-layer network and want to pre-train the first two layers, 1 and 2, then you need a temporary shortcut layer from 2 to the output bypassing 3 and 4.

Unsupervised layerwise pre-training on the other hand has the advantage that, as an unsupervised method, it does not require annotated data. It pre-trains without supervision using denoising autoencoders³. Denoising autoencoders are neural networks whose output is the same as their input. That is, they specialize in developing a robust encoding of an input such that the input can be reconstructed from the encoding alone. The denoising aspect refers to the fact that to encourage robustness, denoising autoencoders are given inputs that have been deliberately corrupted, or “noised” and are expected to reconstruct them without the noise. (Erhan et al., 2010). The noising used in training denoising autoencoders is not the same as the

³ Restricted Boltzmann machines have also been popular for this purpose, but we focus on denoising autoencoders here because they are easier to explain and they have the advantage of being conceptually relevant to using neural networks to normalize noisy text as well (Hinton, 2010).

noise in noisy text, but as we will describe in Section 3.3, the two concepts are not entirely unrelated.

Softmax and Negative Log Likelihood

Neural networks are necessarily built on continuous operations. Due to the nature of backpropagation, anything not differentiable, and therefore anything not continuous, in a neural network cannot be learned. Text, however, is fundamentally discrete, and is therefore a conundrum for continuous-only operations. If one wants to learn how to select the best word for a given scenario, one is making a discrete decision between multiple options, and that cannot be directly differentiated.

The solution to this issue is what is referred to as a soft version of the max function “softmax.” The softmax operation transforms a vector such that each of its values is between zero and one and the new vector sums to one. Mathematically, it is given as:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

Where K is the number of values in the vector. These individual values can alternately be considered posterior probabilities for each of the possible decisions, which can be letters, words, or, in the case of a named entity or part of speech recognition task, tags. To make a selection, one can simply take the highest value in the vector, which corresponds to the most likely option.

To learn using a softmax output function, one can use what is called the negative log likelihood cost function. Negative log likelihood is given mathematically as:

$$- \ln(D(i))$$

Where D is the distribution vector output by the softmax function. We multiply by negative one because the likelihoods are all less than or equal to one and therefore give negative values. The usual standard for cost functions is to have zero as a perfect value and no maximum, so negative log likelihood returns zero when the correct answer is guaranteed by the model and continues to rise as the model’s predicted likelihood of the correct answer gets closer and closer to zero.

Projection Layers

Another related issue with deep learning for text analytics is that words and letters are not easily represented with numbers. Many sources of data are already represented in numbers, such as images, whose underlying red, green, and blue values for each pixel are readily amenable to being converted into 3-dimensional tensors of numbers and sent to a deep neural network. Although text could be given as its underlying numeric representation—a list of ASCII values for each character in a document—the relationship between two letters is not described by the distance between their ASCII values at all in the same way that the relationship between a light and a dark pixel is described by the difference in its brightness value. For this reason, text analytics takes a different approach, the projection layer.

The projection layer is an extension of a simple approach to converting text into numbers—the one-hot representation. The one-hot representation takes a list of discrete inputs, such as the words in a sentence, and builds a vocabulary of all the words in the sentence, assigning an index to each word. The word is not represented as simply the numeric value of its index, however, as this would suggest a false relationship between words. For example, “dog” would have a greater value than “cat” in an alphabetically ordered vocabulary. Instead, the one-hot’s namesake comes from the vector representation it uses for each word. This vector is the length of the vocabulary, and consists of all zeros except for at the index of the word, at which the value is 1. This puts the words in a multidimensional space where each word is on one axis—at full strength in one dimension and at zero in all other dimensions (Ratinov and Turian, 2010).

Although this approach proved effective enough to be a standard approach for a long time (Ratinov and Turian, 2010), it suffered from two major flaws: first, it was extremely sparse – by definition, only one dimension of each vector is actively relevant in a given computation. Since each vector, even zero-valued ones, takes computer time to calculate, this is a problem when trying to run a network efficiently. Second, it permits no concept of semantic similarity. The vector for “cat” is guaranteed to be orthogonal to the vector for “dog,” and if the network is to understand that cats and dogs are both domestic quadrupedal mammals popular around the world, it will have to do so in later layers. The one-hot representation has no ability to make note of such semantic details.

The solution to both of these weaknesses is the projection layer, which allows the network to represent each word, rather than as a point along one axis of vocabulary space, as an arbitrary point in multidimensional semantic space. Instead of being 1 in the dimension of “dog” and 0 in all other dimensions, the word “dog” can have values in many dimensions that it shares with the word “cat” and other words. The projection layer works, like the one-hot method, with a vocabulary that assigns an index to each word. Instead of being a sparse vector the length of the vocabulary, however, the output comes from a matrix of weights into which each word’s index points. Thus, while the output from a one-hot representation layer must be a vector the size of the vocabulary, the output from a projection layer is based on the dimensionality of the semantic space, which is another hyperparameter of the model that can be tuned to fit the problem. The vectors that represent words in semantic space are referred to as semantic embeddings.

The question at this point is how the semantic space is trained to give meaningful values to the word representations. It is often effective simply to allow the projection layer weights to be learned along with the rest of the model via backpropagation, but they can also be learned separately from large, unsupervised corpora using freely available tools such as Word2Vec and GloVe (Mikolov et al., 2013; Pennington and Manning, 2014).

Rectified Linear Units

Mathematical sophistication is not a prerequisite for a nonlinearity in deep learning. Take for example the rectified linear unit. The rectified linear unit speeds computation and can even improve generalization. It accomplishes all this with only a max function.

$$y = \max(x, 0)$$

Thus the rectified linear unit is linear when the input is greater than zero and constant when the input is less than or equal to zero. The rectified linear unit has been successful in a number of natural language tasks such as speech processing and machine translation (Zeiler et al., 2013; Devlin et al., 2014).

3.2 The W-NUT Lexical Normalization for English Tweets Challenge

The Lexical Normalization for English Tweets Challenge associated with the Workshop on Noisy User Text and the Association for Computational Linguistics was held in late 2014, with awards announced in early 2015. The task was split into two sections – constrained and unconstrained, where the constrained subtask disallowed use of external data and the unconstrained subtask did not. Both tasks allowed the use of tools such as ARK-CMU’s Twitter part of speech tagging system (Baldwin et al., 2015).

The dataset is taken from English tweets, constrained to tweets that use all alphanumeric characters. Filtering by language was accomplished with `langid.py` to remove all non-English tweets and tokenization used the CMU-ARK tokenizer (Lui and Baldwin, 2012). To ensure a reasonable proportion of noisy words, the competition organizers excluded tweets that had fewer than two terms each not occurring in their dictionary. The dataset included 5,200 randomly-sampled English tweets which dropped to 4,917 after non-English sentences missed by `langid` were removed. Over the course of the competition, the dataset was corrected twice based on feedback from contestants. Annotators achieved a Cohen’s kappa of 0.5854. Some statistics for the corpus, specifically the number of words corrected by an annotator in the training and test datasets, are given in Table 2. The different columns refer to the change in the number of words with normalization. One to many, for example, refers to a single word before normalization becoming multiple words upon being normalized. (Baldwin et al., 2015).

Table 2: Number of words corrected by an annotator in each dataset

Category	One to one	One to many	Many to one	Overall
Training	2,875	1,043	10	3,928
Test	2,024	704	10	2,738
Training Ratio	0.587	0.597	0.500	0.589

3.3 Architecture and Components

Our model consists of three components: a *Reconstructor* that encodes the input and then reconstructs it in normalized form, a *Flagger* that determines whether the Reconstructor should be used or if the word should be taken as-is, and a *Conformer* that attempts to smooth out

simple errors introduced by quirks in the Reconstructor. Our model does not include any context in its normalization.

For illustration, in this section we will use the simple example transformation of “u” to “you” where “u” is the input text and “you” is the gold standard normalization. In our example we use a maximum word size of three. Figure 3 shows the flow of our example through the model. Information flows from left to right and ellipses represent data objects while rectangles represent processes. In broad overview, the input is preprocessed and sent to both the Reconstructor and the Flagger. The Reconstructor computes a candidate normalization, and the Flagger determines whether to use that candidate or the original word. The Reconstructor’s output is passed to the Conformer, which conforms it to a word in the vocabulary list, and then the candidate, the flag, and the original input word are passed to a simple decision component that either keeps the original word or uses the normalized version based on the output of the Flagger. While it may seem inefficient that the normalized version is always computed, even if it is not used, this approach allows the Reconstructor and Flagger to each run in parallel on many inputs at once.

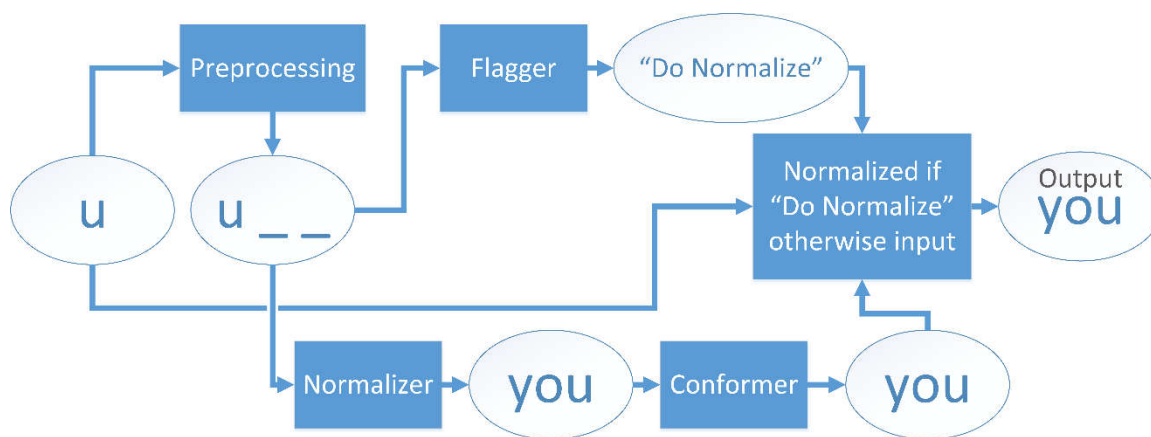


Figure 3: A flowchart detailing the process of normalizing a word

The Reconstructor

Our use of deep feed-forward neural networks for the task of normalization is inspired by the success of denoising autoencoders (Vincent et al., 2008). As we described in Section 3.1,

Denosing autoencoders are neural networks whose output is the same as their input. That is, they specialize in developing a robust encoding of an input such that the input can be reconstructed from the encoding alone. The denosing aspect refers to the fact that to encourage robustness, denosing autoencoders are given inputs that have been deliberately corrupted, or “noised” and are expected to reconstruct them without the noise. It is this “denosing” aspect that makes denosing autoencoders so interesting for lexical normalization.

The main component of our model, the Reconstructor, uses a feed-forward neural network that functions on a similar principle to that of a denosing autoencoder. It reads the character sequence that describes the word and encodes it internally, outputting the denoised (normalized) version. It accomplishes this in three sets of layers. First the character projection layer takes a string and represents it as a fixed-length numeric vector. Next, a feed-forward neural network converts the data into its internal representation and, with a special output layer, into a denoised version of the input. Figure 4 shows a diagram of the Reconstructor’s architecture. The circles represent values, the lines weights.

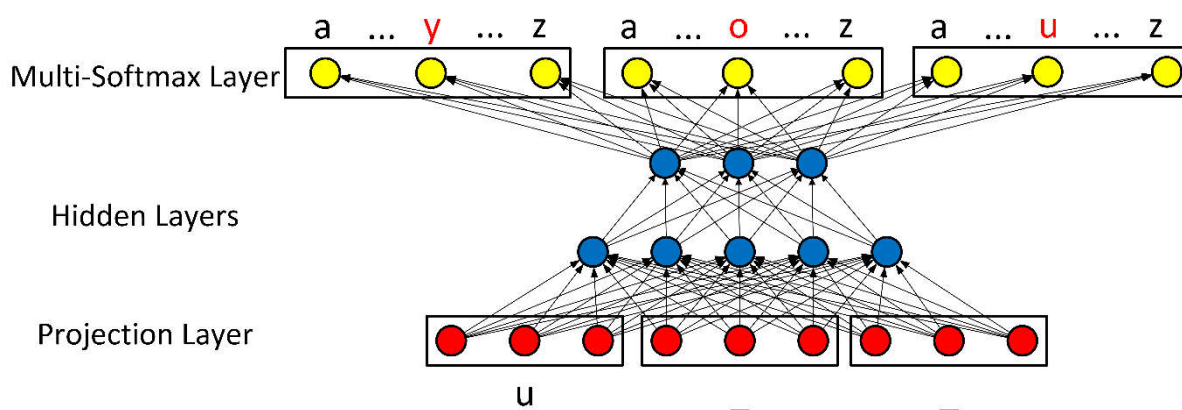


Figure 4: A diagram of the Reconstructor correcting "u" to "you"

The first step of the Reconstructor is performed by the character projection layer (Collobert et al., 2011). The character projection layer learns floating point vector representations of characters, which it concatenates into one large floating point vector word representation. In our example, the letter “u” is represented by n floating point numbers. For example, if $n = 3$ the representation for “u” might be $[0.1, -1.2, -0.3]$. This vector was chosen

arbitrarily, but in the actual model, values are learned in training. The representations allow more information to be associated with a character than a simple numeric index. The character projection layer is described in more detail in Section 3.1.

In this simple example, the word “u” is composed of one character, but if it were longer, each letter would be separately represented. A key challenge at this point is that a feed-forward neural network cannot handle an arbitrary number of inputs. Because each position in the vector is a neuron matched directly to a set of weights, changing the size of the vector would require changing the size of the learned weights, and the model would have to be retrained.

To accommodate this, we use a fixed window. Before we send our input to the Reconstructor, we preprocess it to meet a specified length, filling in unused spaces with a sentinel padding “character” that projects to its own set of learned weights like the other characters. Since the maximum word size in our example is 3, we use a window of size 3. Therefore, our input “u” becomes [u, _, _] and then is projected and concatenated and becomes something like [0.1, -1.2, -0.3, 1.3, 0.0, -1.1, 1.3, 0.0, -1.1]. Notice that we have nine values now in our input. That is the three values from “u” and then the three values for “_” ([1.3, 0.0, -1.1]) twice, once for each “_”. After this step, the system has a numeric vector representation of a word that is always the same length. It now sends it to the first layer of the feed-forward neural network. We deliberately select a large enough window that only in a small minority of cases does a word have to be reduced to fit into the window.

The last hidden layer’s values go through one final matrix multiplication to output a list of values wv in size, where w is the size of the window and v is the number of possible characters including the padding character, that is, the number of characters in the alphabet, which is shared between the input and output layers. In this last layer the nonlinear transformation is a special version of the softmax operation. In this case, we are predicting a window of w characters rather than a single character, so we perform softmax separately on each of the w sets of v values in the layer. In prediction, we simply take the index of the highest value in each of the w sets, but in training we take the whole prediction distribution and try to maximize the likelihood of each correct letter. We do not attempt to predict the character projections in the reconstruction layer because the model is learning them and would be able

to simply minimize its cost function with a trivial function in which all character projections are equal.

Training the Reconstructor as a whole relies on generating posterior distributions and attempting to minimize the total negative log likelihood of the gold standard. Mathematically, our objective function is given as

$$\text{cost} = \sum_{p \in P} \ln(p)$$

Where p is an element in P , the vector of the probabilities of each gold standard letter. So, if our model predicts “y” as 75% likely for character 1, “o” as 95% likely for character 2, and “u” as 89% likely for character 3 in our window of size 3, the negative log likelihoods calculated as (.29, .05, .12) are summed to get the error. This sum error gives a simple measurement of performance to optimize, which backpropagates through the model to learn all the weights described above (Rumelhart et al., 1986).

The Flagger

The Flagger identifies what does and does not require normalization. The vast majority of the training data, 91% in the case of W-NUT, does not require normalization, so returning the reconstructed encoding of every word would risk incorrectly regenerating an already canonical token.

The Flagger has the same general structure as the Reconstructor itself except for the final layer. Instead of generating text at the last layer, a softmax layer predicts whether the token should be normalized at all. Thus, the Flagger’s output layer is two neurons in size, one representing the flag “Do Normalize,” and another representing the flag “Do Not Normalize.” In the construction of the gold standard for the task, there were three reasons a token would not be normalized: firstly, the token is already correct, second, the token is in a protected category (hashtags or foreign words, for example), or third, it was simply unrecognizable such that the human normalizer could not find the correct form. The Flagger accounts for but does not explicitly distinguish between these three possibilities.

The Conformer

Even when a token should be corrected, it is possible that the Reconstructor will come very close to correcting it without succeeding. Reconstructing the word “laughing,” for instance, the Reconstructor can fail completely if it predicts even one letter wrong. An early analysis of W-NUT validation data found that for one set of inputs, the Reconstructor had predicted “laugling” instead of laughing. These off-by-one errors are a frequent enough occurrence to merit a module to deal with them. The Conformer is also useful for correctly normalizing rare words whose correct normalization is too long for the window to represent. In particular “lmfao” expands to an impressive 27 characters, but if the Reconstructor predicts only the first 25 characters, the Conformer can easily select the correct token.

To correct these small Reconstructor errors we construct the Conformer by collecting a dictionary from the gold standard training data. The dictionary is simply a list of all the unique words in the gold standard data. Then at runtime, whenever the Reconstructor runs and predicts a word that is not present in the dictionary, we replace it with the closest word in the dictionary according to Levenshtein distance (Levenshtein, 1966). Ties are resolved based on which word comes first in the dictionary. Because Python’s set function, which does not guarantee a specific order of its contents, is used to construct the dictionary, the dictionary’s order is not predictable and thus ties are resolved unpredictably.

3.4 Settings and Evaluation

The model was implemented in Theano, a Python library for fast evaluation of multi-dimensional arrays using matrix operations (Bastien et al., 2012; Bergstra et al., 2010). We used Theano’s implementation of backpropagation to train our model. For our window size, we selected 25 characters, which is large enough to completely represent 99.9% of the tokens in the training data while remaining computationally feasible. There are also a number of hyperparameters: the number and size of hidden layers, the size of character embeddings, and the dropout rate. We tried various combinations of values between 50 and 6000 for the size and 1 and 4 for the number of hidden layers in both our Reconstructor and Flagger. Some combinations we tried can be seen in the results section. Especially large sizes and numbers of

layers proved to require more memory than our GPU could support, and training them on our CPU was exceptionally slow. We also tried 50% and 75% dropout, meaning that during training we randomly excluded hidden nodes from consideration at each layer. Dropout has been shown to improve performance by discouraging overfitting on the training data, and 50% and 75% are common dropout rates (Srivastava et al., 2014).

We found the highest F1 score on the validation data for the Reconstructor with two hidden layers of size 2000 each and 50% dropout. This was close to the maximum size our GPU could support without reducing the batch size to be too small to take advantage of the parallelism. The Flagger's highest score was found at two hidden layers of size 1000 each and 75% dropout. Attempts to use models wherein the hidden layers were not of matching size consistently found inferior results. For the size of each embedding in the character projection layer, 10 had proven effective earlier in a simpler unpublished Twitter part-of-speech task. We selected 25 for our character embedding size to account for the greater complexity of a normalization task.

We separated the provided training data into 90% training data, 5% validation data and 5% test data. In order to construct a useful model on the small amount of available data, we iterate training over the same data many times. After 150 training iterations, our model was considered to have converged and stopped training. We chose 150 iterations as the smallest value that did not lead to ending the training at a clearly suboptimal value. The training also stops at 5,000 iterations but in practice it always converged before reaching this value.

Early in development we found that the Reconstructor had exceptional trouble reconstructing twitter-specific objects, that is, hash-tags (#goodday), at-mentions (@marysue) and URLs (http://blahblah.com). Generally its behavior in all three cases was to follow the appropriate standard marker character (@, #, http://) with a string of gibberish for the most part unrelated to the word itself. Because these are protected categories that by the terms of the competition should not be changed, we removed them from the training data and rely on the Flagger to flag them as not to be corrected.

We used layer-wise pre-training, meaning we first trained with zero hidden layers (going directly from the character projection to the softmax layer) to initialize the character

embeddings, then we trained with one hidden layer, initializing the character embeddings with their previously trained values. When we trained the full model using two hidden layers, we initialized both the character projection layer and the weights from the projected input to the first hidden layer with the values learned before. The model continued to learn all the weights it used. Pre-trained weights continued to be trained in the full model, although “freezing” some pre-trained weights after pre-training and only training later weights in the full model has shown success when working with large amounts of unsupervised data and may be worthwhile to consider in future work (Yosinski et al., 2014).

Running on an NVIDIA GeForce GTX 680 GPU with 2 GB of onboard memory, training the Reconstructor took about six hours. We do not include CPU and RAM specifications because they were not heavily utilized in the GPU implementation. The Flagger was considerably faster to train than the Reconstructor, taking only a little over half an hour.

3.5 Results and Discussion

The model earned third place in the competition, with scores very close to the second place model. The model’s results in the competition compared to the first, second, and fourth-place models are shown in Table 3. In this table, the Indian Institute of Technology Patna (Iitp) used a CRF-based model. The precision scores are much higher than the recall scores for all models because in this task precision measures the capability of the model to not normalize what does not need normalizing while recall requires that a model both correctly identify what needs to be normalized and correctly normalize it.

In addition to the challenge results, we performed a more in-depth analysis on our own held-out validation and test data. Our analysis of the scores is shown in Table 4. Initial data on the Flagger can be found in Table 5. Our analysis of flagger errors can be found in Table 6. Given the large proportion of errors mistakenly marked “Do Not Normalize,” we looked at these errors. A few examples can be found in Table 7. The “Normalized” column is what the model would have produced if the Flagger had produced the flag “Do Normalize.” Although the Flagger was not trained with Reconstructor confidence in mind, it does an impressive job of only cancelling a normalization when the normalization is either unnecessary or would fail. In no case did the Flagger prevent the Reconstructor from making a correct normalization.

Table 3: Results of the constrained task

Model	Precision	Recall	F1-Score
NCSU_SAS_NING	90.6%	78.7%	84.2%
NCSU_SAS_WOOKHEE	91.4%	74.0%	81.8%
NCSU_SAS_SAM	90.1%	74.4%	81.5%
Term-mapping	93.3%	70.6%	80.4%
Iitp	90.3%	71.9%	80.1%

Table 4: Model scores on validation and test data

Data	Precision	Recall	F1-Score	Accuracy
Validation	89.4%	77.5%	83.1%	97.4%
Test	82.3%	68.7%	74.9%	96.7%

Table 5: Flagger scores on validation and test data

Data	Precision	Recall	F1-Score	Accuracy
Validation	98.2%	99.4%	98.8%	97.8%
Test	97.8%	99.3%	98.6%	97.4%

Table 6: Analysis of errors. Percentages given are out of the total error count

Error	Percentage Occurrence
Correctly flagged, misnormalized	13.85%
Mistakenly flagged “Do Not Normalize”	66.15%
Mistakenly flagged “Do Normalize”	20.00%

Table 7: Examples of tokens that were mistakenly flagged "Do not normalize"

Original	Gold Standard	Normalized
FB	Facebook	fabol
Fuhh	f***	fuhh
OPENFOLLOW	open follow	openffolow
Feela	Feels	feela
Bkuz	because	bkuze
Kin	kind of	kin
Bruuh	brother	bruuhr

An analysis in Figure 5 shows some early results from using only the Reconstructor without a Conformer or Flagger. Model structures are given by “LxN” where L is the size of each layer and N is the number of layers. “More_pre-train” indicates that pre-training has continued for 500 instead of 250 iterations. These cluster at the bottom with the lowest rate of error. To smooth the graphs and make them more interpretable, values at each epoch are the

average of a 10-epoch window. To fit this many runs in a reasonable time span, we used only ten percent of the training data. In this analysis, error rate is measured by token. To put the error rates in perspective, our final error rate was close to three percent. We show this graph to illustrate a number of points. Particularly, we wish to illustrate the challenge of encoding and reconstructing every item in a massive vocabulary, the value of additional iterations of layer-wise pre-training, and the large spikes in the error rates at certain points in the model.

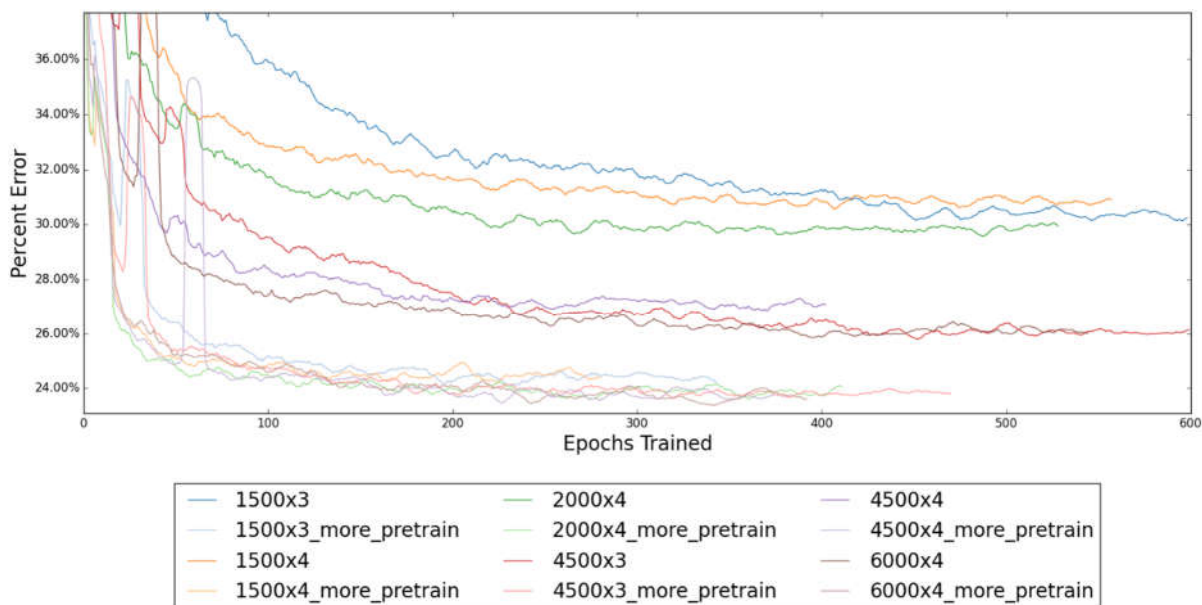


Figure 5: The Reconstructor component validation scores by epoch

The Reconstructor demands much more representational power when not assisted by the Flagger. Before we added the Flagger, we saw continual improvement of results going up to four layers of six thousand nodes each. We saw greater improvements from adding more nodes per layer than from adding more layers. The cluster of three lines near the top of Figure 5 all have layers of 1500 or 2000 nodes each, and the next cluster down is the models we tried with 4500 and 6000 nodes. Incidentally, all but the smallest of these models were too large for our GPU's 2GB of onboard memory, so we ran them over a series of days on the CPU. As a reminder, after we added the flagger, we only required two layers of 2000 nodes each to get competitive results. In each case we used a dropout rate of 50%.

The default models pre-trained each layer for 250 iterations and we also trained models with the same structure for 500 iterations. We find a noticeable improvement in the error rate for the models that were pre-trained for more iterations. In the graph, the models with more pre-training make up the cluster of lines near the bottom of the graph.

It is interesting to observe that some lines have brief spikes multiple percentage points in size. Because it only takes a one-letter mistake for a word to be misnormalized, we expect that at these times a small error arose that affected a large number of words. It is worth pointing out that each model continues to improve while in its spike, eventually dropping back to pre-spike levels.

The model is unique among the three top-performing models in that it avoids external data both directly and through indirect sources. The constrained task does not allow external data, but it does allow the use of off-the-shelf tools trained on external data. Our model does not use any such tools. Without the assistance of tools such as part-of-speech taggers, attempts to use context proved ineffective, likely because of increased sparsity. A given word that appears in the training set three hundred times may only appear three times after another particular word, and may not occur more than once with a particular prior word and following word, so it is more difficult to find patterns in limited data. Future work could either attempt to use tools to provide additional information or could simply take advantage of large amounts of data to learn directly the relationships such tools traditionally abstract for the benefit of conventional machine learning.

3.6 Notes on the Absence of Feature Engineering

One element that we explicitly separate from the act of normalization is the process of identifying whether a word should be normalized in the first place. We do not count this act of flagging errors as a hand-engineered feature because, unlike, for example, the overlap of n-grams between words, it is not an immediately computable quantity. Rather it is a general concept within the umbrella task of lexical normalization that by definition applies to any task under that description. In this chapter, flagging was accomplished with another neural network and the output was used to decide whether to bypass normalization entirely.

3.7 Summary

Normalization of Twitter text is a challenging task. With a direct application of simple deep learning techniques and without relying on any sources of external data, direct or indirect, we built a model that performed competitively with the other models in an international shared task evaluation competition. Our method shows the ability of deep learning to tackle lexical normalization without labor-intensive hand-engineering of features.

To our knowledge, this prior work is the only example of a feed-forward neural network applied to lexical normalization. In its construction we introduce a reconstruction layer for lexical normalization that generates a word character-by-character in a fixed-sized window rather than selecting from a vocabulary. In order to deal with certain weaknesses in the reconstruction layer, we add two additional modules—one that bypasses the reconstruction layer if the word does not require correction and one that selects a term from a vocabulary according to Levenshtein distance to correct for off-by-one errors. This approach focuses entirely on the normalization of Twitter text, but it sets the stage for the task of general lexical normalization.

CHAPTER 4

RELATED WORK: NEURAL MACHINE TRANSLATION

In Section 2.4 we describe how machine translation and noisy lexical normalization are related. Twitter, for example, has frequently been described and even approached as a machine translation task (Kaufmann, 2010; Kobus et al., 2008; Chrupala, 2014). Neural networks, meanwhile, have the ability to learn the relevant features for a given task directly from data, indicating promise for general architectures for use in broad tasks. Therefore, the recent field of neural machine translation (NMT), wherein neural networks are used to translate between human languages, offers an excellent source of inspiration for developing a general normalization architecture.

In this section we will first expand upon Section 3.1's introduction to neural networks in text analytics and discuss some of the advanced neural network systems currently in use in the field in Section 4.1. In Section 4.2 we will describe the state of the literature regarding the combination of machine translation and neural networks, neural machine translation, and finally in Section 4.3 we will discuss how the task of lexical normalization is similar to and different from that of neural machine translation and what it means for our work.

4.1 Advanced Neural Network Structures

The structures discussed in this section are designed to account for the arbitrary-size inputs endemic to natural language processing. Whereas a typical neural network expects an input of a fixed size, this is an absurd restriction to place on human words and sentences that they be a fixed size. Recurrent neural networks and convolutional neural networks both address this by taking a single feedforward network and applying it an arbitrary number of times to the data. In this section we will describe the differences between these two types of advanced neural networks.

Recurrent Neural Networks

A recurrent neural network (RNN) is among the simplest examples of a network that does not have to have a large number of layers to be deep. In a recursive neural network, the depth of

representation for which deep learning is known is achieved through repeated application of the recurrent layer. This notion of repeated application is what makes recurrent neural networks naturally amenable to the arbitrary length of words and sentences. The recurrent layer is structured much in the same way as any layer in a conventional feed-forward network; the difference is that one of its inputs is itself (LeCun et al., 2015). See Figure 6 for a diagram of a simple recurrent neural network run on the first two words of the sentence “Child born to Duchess of Cambridge.” Each circle represents a node and lines represent the flow of information. The network visits each word in order, so at the first timestep ($t1$) the network visits the first word of the sentence, then at $t2$ it visits the second word, etcetera. Notice especially that Hidden State 1 functions both as a hidden state and as the input at the next timestep. Hidden State 0 is arbitrarily initialized, often as all zeros.

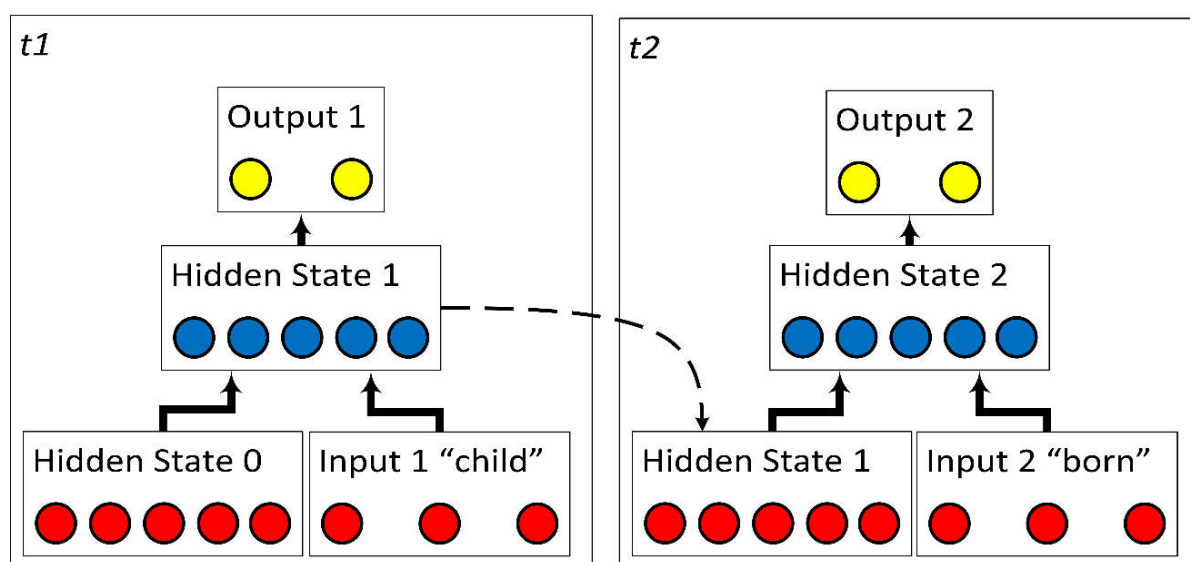


Figure 6: A diagram of a simple recurrent neural network

Recurrent neural networks take a number of forms. The basic recurrent neural network is often referred to as a simple recurrent network (SRN) (Chrupala, 2014), while another method, the long short-term memory network (LSTM), uses adaptive gates that modify the rate at which certain neurons in the hidden layer change. These adaptive gates correct for a known weakness

of the SRN that it is less effective at remembering less recent inputs, but comes at the cost of increased computational complexity and complexity of implementation (LeCun et al., 2015).

The LSTM belongs to a set of networks known as gated recurrent neural networks. Another gated recurrent neural network that attempts to achieve the memory of the LSTM without its complexity uses, rather than the four gates of the LSTM, what is referred to as a gated recurrent unit (GRU) with only two gates to help it remember distant inputs. Despite having fewer parameters to learn, the performance of a gated RNN is comparable to that of an LSTM (Chung et al., 2014). Figure 7 shows how the GRU changes how the hidden layer and output are calculated. For each node of the hidden layer, the reset gate z and update gate r adaptively manage the degree to which new knowledge should rely on old and the speed at which old knowledge is replaced with new respectively (Chung et al., 2014).

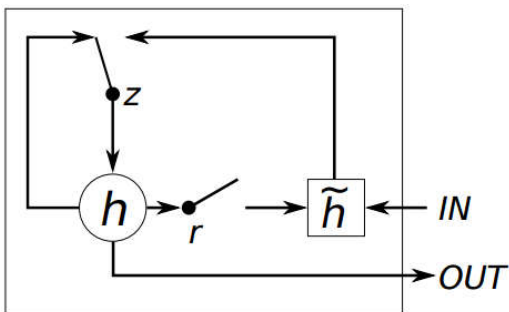


Figure 7: The gated recurrent unit

In more technical terms, the reset gate r places a weight on the influence of the previous hidden layer on the candidate new hidden layer \tilde{h} between 0 and 1 where 0 means it has no influence, and 1 gives it the full influence it would have in an SRN. In the event that previous inputs are no longer important, it can be valuable to be able to forget them. For instance, one would not need to remember the count of a noun if one is sure that one has already seen all the other words that would need to agree with it. Other than the reset gate, the candidate new hidden layer is computed in the same way as the new hidden layer is for an SRN. The candidate new hidden layer is only an intermediate value, however, and the real new hidden layer is calculated via interpolation between the candidate hidden layer and the previous hidden layer managed by the reset gate. Here, a reset gate value of zero means that the new hidden layer is

the same as the old hidden layer with no influence from the candidate and 1 mimics the SRN, completely losing the old hidden layer in favor of the new. Keep in mind that although we refer to layers here, in fact each node within each layer is calculated separately, meaning one element of knowledge can carry throughout the entire sentence while another changes with every word. This is key to the effectiveness of the GRU and the gated RNN. When we refer to a gated recurrent neural network in this dissertation, ours uses a GRU.

The last important alternative type of the recurrent neural network that we will cover here is the bidirectional recurrent neural network (BiRNN). The BiRNN permits the RNN to accept context from both directions. This means that “cat” in “the cat sat” will benefit from both “the” and “sat” rather than just “the” as with a typical forward-only RNN. The BiRNN’s output is calculated from the concatenated hidden layers of two RNNs that traverse the data in opposite directions (Schuster and Paliwal, 1997). Bidirectionality and gating can both be applied to the same network to make a bidirectional gated recurrent neural network, or the “gated BiRNN.”

Convolutional Neural Networks

Another popular technique for the analysis of text input is the convolutional neural network (CNN). CNNs were originally designed for image analysis, and remain effective for this task (Vinyals et al., 2015). It is only recently that this approach has gained traction in text analytics as well. A convolutional neural network is defined by the presence of a convolutional layer, which, like a recurrent layer, uses repeated application of a fixed-size set of weights, known as a filter, to collect a set of features for each position within an input. Instead of moving linearly through the input like an RNN, a CNN’s convolution layer separates the input into windows of a fixed size and performs its transformation simultaneously on every window. The convolutional layer is matched with a K-max pooling layer that makes the arbitrary-size input manageable by selecting the K highest values from the features.

Convolution and max pooling work differently for text than for images. Firstly, an image is two-dimensional, so it uses what is called a 2D convolution, whereas text is what is known as a time sequence, with one word appearing after another in one dimension and therefore uses a 1D convolution. We will not describe a 2D convolution in detail here. The 1D convolution uses a 1D filter that acts as a feed-forward layer on each window of a given input, so if we had

an input headline of “Child Born to Duchess of Cambridge,” and a filter of size 3 we would operate on “Child born to” “Born to Duchess” “to Duchess of” and “Duchess of Cambridge.” We can also place dummy values on either side of the sentence to make sure that the words on the ends are not underrepresented. See Figure 8 for details. On top is a narrow convolution where “Child,” “Born,” “of,” and “Cambridge” on the edges are all underrepresented compared to “to” and “Duchess” in the center. The wide convolution on bottom resolves this by adding padding inputs on either side.

Each of these windowed inputs would go through the filter and output a value. To 1D convolve over a multidimensional semantic embedding representation, we convolve separately over each feature within the semantic space, so for a sentence with N windows and an embedding of size E we end up with a hidden matrix representation of size $N \times E$.

K-Max pooling over this multi-feature representation similarly takes each feature separately. The operation is straightforward except that the K values remain in the order in which they first appeared. This operation takes the hidden matrix of size $N \times E$ and returns one of $K \times E$. This matrix is then flattened into a layer of KE features, which since K and E are both fixed with respect to the input, is appropriate to send to an FFNN or other neural network (Kalchbrenner et al., 2014).

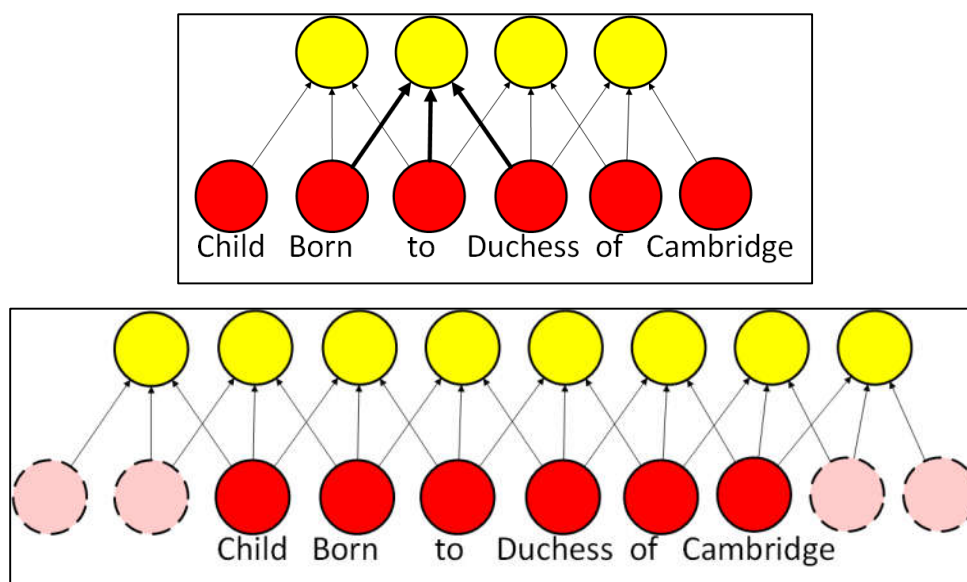


Figure 8: A narrow (top) and wide (bottom) convolution for a given headline

4.2 The History of Neural Machine Translation

Neural machine translation (NMT) arises naturally from the idea that a neural network is capable of representing data in an internal form from which it can be reconstructed. One can therefore achieve translation by encoding text from one language and decoding it into another language. This is the basis for NMT. In this section we will describe the papers that have been published in NMT since 2013 to emphasize the field's nascence and impact and set the stage for us to highlight its relevance to our normalization framework in later sections.

Because they are naturally amenable to the arbitrary-length input that defines text, convolutional and recurrent neural networks form the basis of NMT. Kalchbrenner & Blunsom (2013) use a combination of these two layer types. Their technique, the recurrent continuous translation model (RCTM), relies on what they call a convolutional sentence model to develop an internal representation of a sentence and a recurrent neural network to then generate a new representation in a different language. This approach of encoding the sentence into a fixed-length vector and decoding it again is known as the encoder-decoder approach to neural machine normalization. In the same work, they propose a preliminary alternative model in which the convolutional network encodes the input and then an inverted convolutional network decodes it to length m , the length of the target sentence such that there is a separate internal representation for each step of the RNN that generates the output sentence. The results from this model are superior to those of the encoder-decoder approach, but in their evaluation, Kalchbrenner and Blunsom set m for each sentence to equal the length of the gold standard target sentence, making this approach impractical without an external means of estimating m .

After Kalchbrenner and Blunsom's work, two groups separately (Cho et al., 2014b; Ilya Sutskever, Oriol Vinyals, 2014) proposed models that used RNNs for both encoding and decoding. One group used LSTMs while the other took used the gated recurrent unit. In the same work the same group proposed a recursive gated convolutional network that can adaptively choose its own structure by selectively opening and closing gates to join or separate nearby concepts. We show their diagram of some of their advancements in Figure 9. In Figure 9, (a) is a recurrent convolutional neural network. Open circles are input words and closed circles are hidden phrase representations. The arrows represent connections that can be opened

or closed to represent different phrase structures. Also in Figure 9, (b) is the gated activation function. The dotted and solid arrows represent the flow of information and the diagonal line represents the gate that decides whether the first input or the second input or a neural combination of the two should be propagated to the next layer. Two examples of ways in which nodes can be combined by a recursive gated convolutional neural network are shown in (c) and (d). The arrows represent where information from a node has been included in the calculation of the following node (Cho et al., 2014a).

In 2015, the encoder-decoder approach was applied to the automatic captioning of images while a competing approach outperformed it in machine translation. Using a deep vision CNN and a language-generating LSTM, a group from Google developed a system that automatically produces captions for images (see Figure 10). (Vinyals et al., 2015). This system shows the versatility of the encoder-decoder approach.

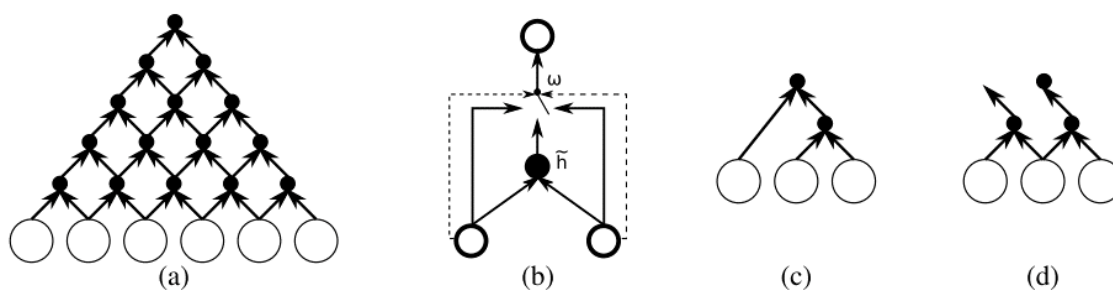


Figure 9: The recursive gated convolutional network

Absent in previous approaches, the ability to translate sentences over thirty words long without suffering reduction in quality came from an entirely new approach. Bahdanau, Cho, & Bengio (2015) develop a model to automatically search for the sections of the source text that are most meaningful to the target word to be translated. The model involves a number of steps which we will describe one at a time. First, a gated bidirectional RNN provides information about the words preceding and the words following the word at each time-step i . This information about a word and its context at a given time-step is referred to as an annotation,

and in a sentence of length T there will be T annotations. These annotations are then used to inform the RNN that generates the translation. It is the presence of multiple annotations to represent a sentence instead of just one fixed-length vector that allows this model to generalize to translate longer sentences.

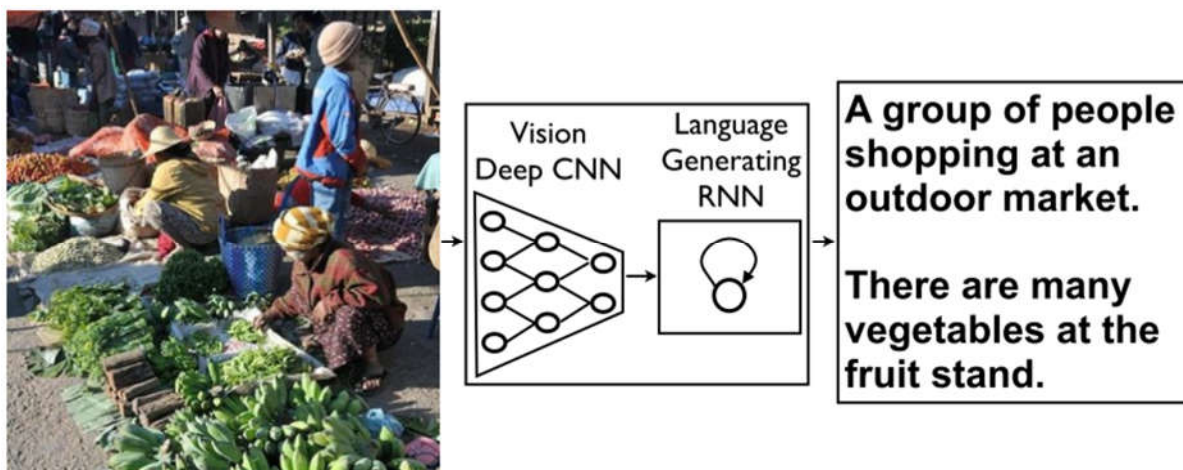


Figure 10: An automatic caption generator using NMT techniques

The novel layer that makes this approach successful is a special type of attention-based model that allows the network to weight different annotations based on their relevance and make a fuzzy selection of an arbitrary number of these annotations. At each time-step in the generation of the target sentence, the network generates relevance scores for each annotation using a feed-forward neural network. The network is informed by the content of the annotation at position i in the source text and the previous state of the generative neural network (S_{t-1}). The T relevance scores are transformed via a softmax function (described in Section 3.1) and then the context vector that will inform the generative recurrent network for its next word is a weighted average of all the annotations by their respective relevance scores (Bahdanau et al., 2015). Figure 11 shows a diagram of the annotation-based neural machine translation system. $a_{t,i}$ represents the annotation weight at target-time t and source-time i . h_i is the source's bidirectional RNN's hidden layer at source-time i and s_t is the hidden layer of the generative RNN at target-time t . y_t is the output word at target-time t (Bahdanau et al., 2015).

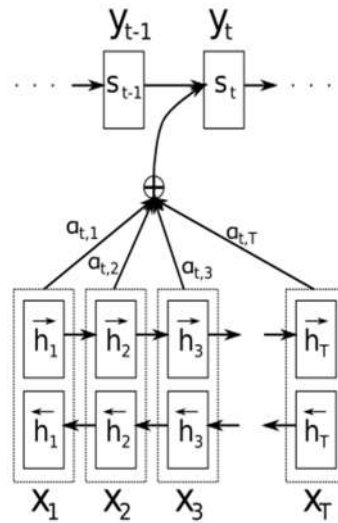


Figure 11: The annotation-based neural machine translation system

Out-of-Vocabulary words

One concern that has received much specific attention in neural machine translation and is relevant to neural lexical normalization is out-of-vocabulary (OOV) words. In a word-based model, it is clearly very difficult to do anything with a word that one has never seen before, but even in a character-based normalization model it is difficult to either recognize or correct to a word that the system does not know (Luong, 2015; Jean et al., 2015; Leeman-Munk et al., 2015).

The obvious answer is to have as comprehensive a dictionary as possible, but this presents a problem for neural networks. A common output for a neural translation model is a probability distribution for a given target word w over a given vocabulary V . Because a probability must be generated for each word in the vocabulary, time and space complexity are proportional to V . For this reason, vocabulary size is frequently limited to what is called a *shortlist*, a target vocabulary of the K most common words. K in this case generally ranges from 30,000 to 80,000 (Jean et al., 2015). For a sense of scale, the Oxford English Dictionary (the OED) as of 2012, included 171,476 words currently in use. Note here that the OED does not include inflections, esoteric technical jargon, or proper nouns (Bassil and Semaan, 2012).

A common solution for when a word does not appear in a shortlist is to give it a special term <UNK> that is treated as a word in and of itself. While this is effective when there are only a small number of such terms in a sentence, it has been observed that the quality of the translation degrades precipitously as more and more unknown words are encountered (Jean et al., 2015). Another approach is to identify the words that are not translatable and place them as-is in the final sentence. That is, if this system were translating the French sentence “Je suis le grand Muzzy” and did not have in its vocabulary the name of the fictional character “Muzzy,” it would still be able to return “I am the great Muzzy.”(Luong, 2015).

Approaches to lessen the burden of a very large vocabulary exist, but no one approach has come out as a clear superior. Bengio & Sen cal (2008) use a less accurate but more efficient model to provide a reduced set of possibilities for the word being selected. Then at each prediction the correct word and these alternative possibilities are the only words that need to be considered. They use an n-gram model and report a training speed increase of 150 times with a negligible change in perplexity compared with a conventional approach. Jean et al. (2015) show that this can work for neural machine translation in particular. The main drawback of this approach is that it requires two models – the neural model and another model that can run much more quickly and still perform effectively. Another approach is in training to use noise contrastive estimation, stochastically selecting one or more other words at random from a large vocabulary to take as negative examples. This approach has been very effective in the efficient estimation of the word vectors that have become ubiquitous in recent years (Mnih and Kavukcuoglu, 2013; Mikolov et al., 2013; Gutmann and Hyv rinen, 2010). This approach is useful at train time, but less so at test time when the correct word is not available to compare to a random word and again the system must compute a probability for every vocabulary word. This can be a serious problem in a real-time system.

4.3 Neural Lexical Normalization

NMT is a useful source of inspiration for neural machine normalization because each is a task of converting unstructured text information of one type into unstructured text of another type while preserving meaning. The first and most important thing neural machine translation has to offer text normalization is a robust approach to handling context. Approaches to handling a

very large vocabulary are also helpful, as this issue is largely the same between text normalization and machine translation. Finally, and most importantly, none of the approaches for neural machine translation thus far are sufficient for lexical normalization because they ignore morphology, a key aspect of normalizing text.

NMT systems are inherently context-based. Encoding-decoding systems take all the information of the input and store it in one fixed-length vector and then generate a new sentence from that input while the alignment-based system adaptively selects the most relevant sections of the input to inform its output. In each case, the entire input sentence is considered when selecting each word in the output sentence. Since the correct form of a normalized word is also based on context, these architectures will be valuable to lexical normalization.

When using NMT approaches for context, we must keep in mind that NMT assumes that ordering may change frequently, whereas in normalization ordering changes much more rarely, so rarely in fact that our task assumes that ordering will not change. Rather than translating between entirely different languages with potentially unrelated grammatical structure, we are translating merely between different forms of a single language. While the structure is not always maintained, it will generally be very similar or the same. Another aspect of translating between dialects of a language is that there is frequently a clearer single normalization for a given noisy text than there is a single translation for a text in a given language. Greater precision is required for NTN than NMT, for instance normalizing “bg guy” to “large man” would be considered incorrect in normalization even if it maintains the meaning well enough to pass a translation test.

Finally, the key weakness of the above neural network architectures for lexical normalization is that they operate at the word level. A key issue of normalization is spelling correction. In a completely word-based model on noisy text, vocabulary grows out of control and morphological noising patterns are lost. In order to be effective for normalization, any neural translation-based approach must include morphological analysis.

It should be noted that although these techniques have not yet been applied to lexical normalization, the area of character-aware neural language models is of significant current interest in the NLP community. As recently as August 2015, Kim, Jernite, Sontag, & Rush

(2015) reported encouraging new results on the generation of semantic representations of words from a combination of their context and their morphological state. Kim, Jernite, Sontag, & Rush explicitly state that their work could be applied to the normalization of noisy text, but they stop short of actually applying it.

CHAPTER 5

MORPHOSYNTACTIC NEURAL NETWORKS FOR GENERALIZED TEXT NORMALIZATION

The work presented in Chapter 3 demonstrates that deep learning techniques can be competitive with the state-of-the-art for Twitter lexical normalization. We have also shown that many of the challenges in optical character recognition post-processing and automatic speech recognition post-processing, such as missing characters and phonetic spelling substitutions, are also present in Twitter lexical normalization. Nevertheless previous work has treated these as three separate tasks, developing different sets of features for each, and there is no example in the literature where a single approach has been evaluated on multiple such tasks.

The advent of deep learning brings with it the ability for a system not only to learn a task based on a set of features, but how to learn the features themselves. This gives an unprecedented opportunity for generalization. Deep learning has also revolutionized the field of text analytics, especially offering new approaches to machine translation, a field often compared to lexical normalization.

With a combination of neural machine translation techniques and morphological analysis techniques explored in our prior work, we have developed an architecture, DeepNorm, for general lexical normalization that can function effectively on text from multiple noisy sources with the expectation that it will be able to generalize to new sources from data without the need for additional expensive engineering of features. In this chapter we describe the development and evaluation of this approach. We first describe our new architecture in Section 5.1, then in Section 5.2 we describe our corpora, followed by our evaluation process in Section 5.3, results in Section 5.4 and discussion in Section 5.5.

5.1 Architecture

In order to be applicable to diverse text normalization tasks, our model eschews any hand-engineered features not trivially derivable directly from the task definition itself. As before, our model’s normalization approach is based on two predictions: a Flagger identifies whether a word needs to be normalized or should be left as-is, and a Reconstructor predicts the correct

version of the word. The improvements in this model are a new morphological analysis approach and the addition of a new layer to include the surrounding words as input when normalizing.

We will discuss each module of our system in order from input to output (Figure 12). The first step is preprocessing (Section 3.1) followed by the generation of noisy word embeddings (Section 3.2), then word-level contextual analysis (Section 3.3), followed by the generation of output normal words (Section 3.4). We will use the simple example transformation of “yu” to “you” where “yu” is the input text, and “you” is the gold standard normalized output.

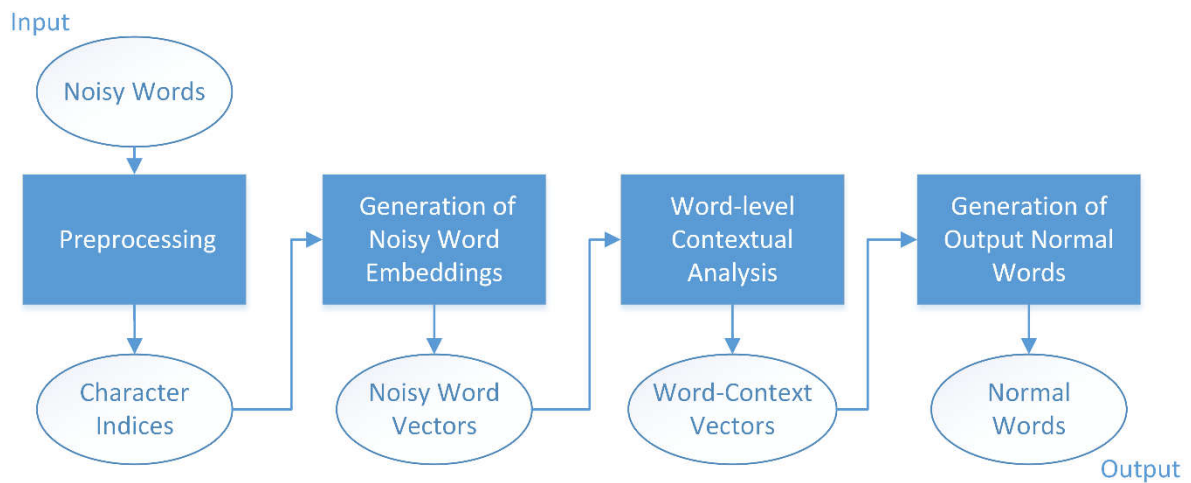


Figure 12: Flow of data through DeepNorm

Preprocessing

As before, the first step of the normalizer is to convert a word, a variable-length list of characters, into a vector, a fixed-length list of floating-point numbers. Once this conversion has been performed, vectors will represent the word until the last step, at which point the word will be reconstructed.

Again, we use a fixed window size in our model, which must be large enough to accept long words. A change from our earlier design, in order to give equal representation to both the beginning and the end of the word, we place the word at both the beginning and the end of the

window, giving the beginning side preference when the word is more than half as long as the window. Thus, in our example of “yu” and using an unrealistically small input window size of 3 for simplicity, our input is converted to the following window:

y	u	u
---	---	---

Here we see that position 0 is guaranteed to be the first letter of the word “y” while as long as the word is smaller than the window, the last position in the window is guaranteed to be the last letter of the word, “u”. This is important because each window position in a feedforward neural network is represented as a feature. If the neural network always has the last letter of the word in the same position it can more easily make decisions based on what the last letter of the word is than if it has to learn at higher layers that the last letter of the word is the one that immediately precedes the padding and might appear at any location in the window. The “y” of the second “yu” is not written because the “u” of the first takes precedent.

This window is then converted into integers. Each integer represents the index of the given letter into a provided alphabet. The provided alphabet is drawn from the training data and may include capital and lowercase letters and punctuation and special characters as necessary. The first character of the alphabet at position zero is a sentinel value that indicates unused space. For example, if we had a window size of five, our example would be as follows:

25	21	0	25	21
----	----	---	----	----

With the example window size of 3, we have the following:

25	21	21
----	----	----

Generation of Noisy Word Embeddings

In this step DeepNorm is largely identical to the prior architecture. It takes the list of character indices generated in Section 3.1 and converts it into an abstract numeric representation, or an

“embedding.” this uses a character projection layer (Collobert et al., 2011). The character projection layer learns floating point vector representations of characters, which it concatenates into one large floating point vector word representation.

The resulting vector is sent to a feed-forward neural network (FFNN), a sequence of one or more feed-forward layers, each of which is a linear transformation followed by a nonlinear function. The output is an embedding for the given word.

Word-level Contextual Analysis

Now that DeepNorm has generated embeddings for each noisy word, it arranges them into a sentence. For our example, we will say “yu” occurs in the sentence “yu think ur so smart.” Much the same as in the generation of noisy word embeddings we have a list of indices. If our vocabulary is in alphabetical order: “smart”, “so”, ”think”, “ur” , “yu”, we represent our sentence as follows:

5	3	4	2	1
---	---	---	---	---

The vector our system generated for the word “yu” is now the first vector in a sentence matrix. The technique we use here is the gated bidirectional recurrent neural network (gated BiRNN) for analysis of context. This technique is similar to other BiRNNs used for context in recent text analytics systems (Kim et al., 2015; Bahdanau et al., 2015). The gated BiRNN reads the sentence up to the word and the sentence backwards from the end to the word and concatenates the two resulting representations (Pezeshki, 2015). Now “yu” and each other word is represented by the concatenated hidden layer at its position in the sentence. This hidden layer goes to another FFNN similar to that described in Section 3.2 and generates another abstract representation of each word and its context in the sentence.

Generation of Output Normal Words

To generate the normalized word, we use a unified multi-softmax layer that predicts both the Reconstructor and the Flagger in one layer. Illustrated in Figure 13, our multi-softmax layer

uses a final feed-forward layer to generate a prediction of the most likely character in each position of the output window as well as a flag indicating whether the word should be normalized or left as-is. The multi-softmax layer does this by generating a list of values $a(w + 1)$ in size, where w is the size of the output window, not necessarily the same size as the input window, and a is the alphabet size. All but the $+1$ of this output is the Reconstructor. The $+1$ comes from the simultaneous Flagger prediction. The Flagger is given a full set of a options, but it only uses the first two, “YES” (do normalize) and “NO” (do not normalize). This allows it to be another row in the matrix multiplication rather than a separate operation. The details of how a multi-softmax layer works in general are described in Section 3.3.

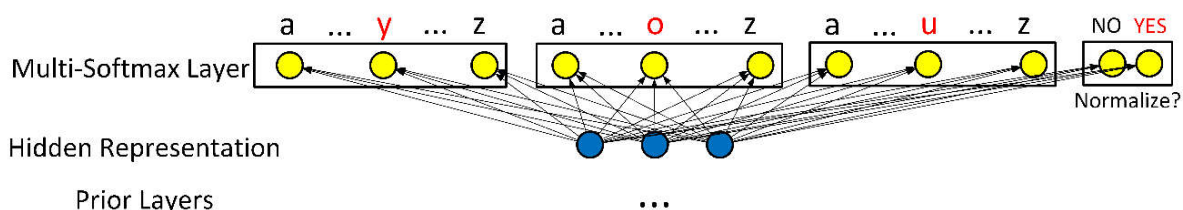


Figure 13: The unified multi-softmax output layer

Once the multi-softmax layer has generated its distributions, we take the predicted letters, separate the Flagger prediction, strip off padding space, and return either the original word or the normalized word depending on the output of the Flagger. In the case of the example, “you” would be the output.

5.2 Corpora

In order to determine the usefulness of our general lexical normalizer, we must evaluate it on multiple corpora. For this purpose we have collected three corpora, an optical character recognition corpus, a Twitter corpus, and an automatic speech recognition corpus. For the former two we have also collected specialized normalizers. We expect our general normalizer to perform competitively against the specialized normalizers on their own corpora and to outperform significantly on corpora for which they are not specialized. In this section we introduce the three corpora on which we evaluate DeepNorm.

Twitter Lexical Normalization (W-NUT)

The W-NUT training data, the same dataset as in Chapter 3, consists of Twitter data aligned with its normalization. W-NUT was officially a Lexical Normalization task and ignored grammatical errors. For simplicity the task defines punctuation, emoji, and tweetcode (@attention, #hashtag, http://url.com) as to be left as-is. As part of the rules of the task, normalization output is guaranteed lowercase, including for words that are otherwise not normalized. In the W-NUT competition, each observation consists of one word in the context of a tweet. The gold-standard for the training and test sets is its normalization as given by a human normalizer, which may be one or more words. In cases where multiple words are reduced to one word in normalization (e.g., “l o v e” becoming “love”) the resulting empty spaces are represented as empty strings associated with the dropped words, but these cases account for only 0.02% of the corpus. The training dataset contains 44,385 words, and the test dataset has 29,421.

Optical Character Recognition Post-processing (Holmes)

For evaluation of our model as applied to optical character recognition post-processing, we use an in-house OCR error corpus we refer to as *Holmes*. Our corpus comes from a transcription of *The Adventures of Sherlock Holmes* (Doyle, 1892) and is degraded by rendering the text in 18pt font, saving as a JPEG with 50% quality, and thresholding at 50% brightness such that all pixels below 50% brightness turn black and all above turn white⁴. The transcription comes from Project Gutenberg⁵. The results of this process, magnified for detail, are shown in Figure 14. Although the degradation looks slight to a human eye, it is enough to generate close to ten percent character error rate. Because OCR errors vary widely based on factors such as lighting, font, and age of the document, our technique is less intended to mimic a particular OCR challenge than to represent OCR errors in general. The errors arising from this degradation can be seen later in Table 14.

⁴ This image processing was accomplished using an ImageMagick wrapper for Python: <https://pypi.python.org/pypi/Wand>

⁵ <http://www.gutenberg.org/cache/epub/1661/pg1661.txt>

We OCRed the text using the open-source OCR software *Tesseract*⁶ tokenized using Carnegie Mellon’s tweet tokenizer *Twokenize*⁷ and aligned using a Levenshtein distance-based Viterbi algorithm. Twokenize was chosen to make the Holmes task more analogous to W-NUT in how it splits punctuation from words. The W-NUT task gives each punctuation mark its own token (when the punctuation is not part of emoji) without breaking words up, whereas splitting on spaces would associate punctuation marks with words and the Python NLTK library splits the possessive “’s” from words.



The light in our window

Figure 14: Text from the Holmes corpus after degradation

Some of the differences between Holmes and W-NUT we decided to keep. For example, Tesseract frequently missed periods at the end of sentences, and as such restoring these periods became another part of the task. We keep Holmes case-sensitive rather than converting all output to lowercase as had been the decision for W-NUT. We made the decisions to keep the differences in order to better represent the task of optical character recognition post-processing. For evaluation on Holmes, we modified Skip-Bigram’s source code not to assume lowercase, as it had been designed to do for W-NUT.

In order to maintain a fair comparison with W-NUT, we use approximately the same number of words of training data, 45,996. In order to achieve a more accurate test result we increased our test set size to 60,001 words.

Automatic Speech Recognition Post-processing (Librispeech)

We generate our automatic speech recognition corpus from Librispeech, a speech recognition training corpus consisting of audio from the public domain collection of readings of classic

⁶ <https://github.com/tesseract-ocr>

⁷ <https://github.com/myleott/ark-twokenize-py>

novels Librivox paired with the text from the appropriate section of the book. Librispeech includes over 500 hours' worth of data, but we selected the dev-clean corpus, in which there are 2,703 transcriptions comprising 54,402 words in total. The median transcription length is 17 words, and the average is slightly above 20. To generate ASR error noise in our corpus we used CMU-Sphinx to recognize the speech of the readers. Without any additional degradation, the output text was noisy enough to make this a challenging task. In fact, it features some of the most challenging corrections of any of the tasks. See Table 8 for a few examples of the sentences generated by CMU-Sphinx.

Table 8: CMU-Sphinx recognitions of speech (top row) as aligned to the ground truth text (bottom row)

and	us	anymore	not	even	a	Death	threat	rabbi	guys	will	last	forever
nothing can part	us	any more	not	even		Death	for	love	like	ours will	last	forever

n.'s	carl	youngster	'who	was	Heard	and	applicable	to	his	last
it was	karl	yundt	'who	was	Heard		implacable	to	his	breath

in	new	their	initial	Warming	your	read	as	church
he had	neither	a	national	Army	nor	an	organized	church

and	there	is	an	old	story	About	this	which	i	shall	tell	you
and	there	is	an	old	story	About	this	which	i	shall	tell	you

The major issue with post-processing of automatic speech recognition errors is that the ASR system has an internal language model that 1) always outputs real words, making all errors real-word errors, and thereby 2) takes output further from its phonetically-based erroneous state to an even more erroneous state. The noised text is not once but twice removed in complex and unpredictable ways from the input.

5.3 Evaluation

DeepNorm is implemented in Theano, a Python library for fast evaluation of multi-dimensional arrays using matrix operations (Bastien et al., 2012; Bergstra et al., 2010). We used Theano’s implementation of backpropagation to train our model. For our input window size, we selected 40 characters, which is large enough to completely represent 100% of the tokens in the training data. Our output window is smaller, only 27 characters based on the expansion of “lmfao” because the longest noisy words were just garbage that could be flagged to leave as-is (the official instruction for human normalizers in W-NUT).

We separated the training data into 95% training data and 5% validation data and selected hyperparameters according to what achieved the lowest error rate on the validation data. Our official evaluation is based on a third test set not used in training. Based on our most successful combinations from Chapter 3, we tried 1 and 2 for the number of hidden layers leading up to the word embedding and again for the recurrent layer in the gated BiRNN and the FFNN after. In each case we tried 1,600 and 3,200 for the number of nodes per layer. We ended up using two layers of 3,200 nodes each leading up to the BiRNN, 800 in each direction for the hidden layer in the BiRNN, and one layer of 1,600 nodes after the BiRNN. We used 50% dropout, meaning that during training we randomly excluded hidden nodes from computation at each layer. Dropout has been shown to improve performance by discouraging overfitting on the training data, and a rate of 50% is well-represented in the literature (Hinton, 2014).

We train our model for multiple iterations over the training data until an examination of the performance graph suggests that no further improvement may be had by continuing training. We found this happened typically between 500 and 1,000 iterations after pre-training. An example of such a performance graph can be seen in Figure 4. At this point, we use the neural network weights at the epoch with the lowest character error rate on the validation set. Running on a Tesla K20m GPU with 5 GB of onboard memory, training our model takes on the order of ten hours.

To pre-train our model we trained first by word for five hundred epochs and then, using the same weights, trained again with context. That is, we used a version of DeepNorm that skips the transformations described in Section 3.3 and uses a single feed-forward network to

go directly to the output layer from the initial word embeddings. We select five hundred epochs as an estimate based on looking at performance graphs of how long it will take the pre-training to converge. Layerwise pre-training is common in deep learning and allows a model to have a more meaningful internal representation of data (Le et al., 2011). In our results we include an alternate version of DeepNorm that stops training before switching to “by sentence.” We distinguish this from our main DeepNorm model, which operates at the sentence level by referring to it as DeepNorm by Word.

DeepNorm’s single task-specific hand-engineered feature is to include the official post-processing steps explicitly used in the construction of a task’s corpus in generating the Flagger training data which keeps track of what words are normalized and what should be kept as-is. That is, for W-NUT, the flagger considers whether a word requires normalization on a case insensitive basis just as the task itself does, while for Holmes it counts case as a difference. As described above, we make sure that our competing models are similarly given the benefit of task-defined post-processing steps.

5.4 Results

Although each specialized model is the best in the task for which it was designed, our generalized lexical normalization model DeepNorm, specifically the version that uses context, DeepNorm by Sentence, significantly outperforms each of its competitors on the data for which the competitor was not specialized. Unlike its competitors, DeepNorm includes no task-specific expert-engineered features designed to maximize performance on a particular source of noisy text such as the character alignment in CLSTM and the fuzzy dictionary lookup in Skip-Bigram. Significance testing is by one-way approximate randomization ($R=10,000$) against the next lowest error rate on the same task (*: $p<0.05$, **: $p<0.001$). (Yeh, 2000). As DeepNorm by Word and DeepNorm by Sentence have different architectures, we are obliged to choose one to represent our generalized architecture. For this purpose we select DeepNorm by Sentence. As our selected architecture, we test its significance against Skip-Bigram on W-NUT and CLSTM by Sentence on Holmes in Table 9. CLSTM by sentence is not included in Table 10, Table 11, and Table 12 because it does not operate at the word level and thus cannot be measured according to its word outputs. Figure 15 shows how similar training is on both

datasets. W-NUT shows a faster convergence by sentence, which fits with the other evidence that it gains little benefit from inclusion of context. It also learns very slowly at first by word, which may be related to the Flagger’s tendency to start off by marking everything as “Do Not Normalize” and stay there for several epochs.

Table 9: Evaluation results comparing character error rate between models

Model	W-NUT	Holmes	Librispeech
Preprocessing Only	6.59%	9.46%	20.25%
Term Mapping	1.59%**	5.39%**	23.45%**
DeepNorm by Sentence	1.45%**	2.69%**	20.25% ^{not sig}
DeepNorm by Word	1.36%*	4.64%**	20.25%**
Skip-Bigram	1.2%**	5.1%**	25.73%
CLSTM by Sentence	5.27%**	2.4%**	20.18% ^{not sig}
CLSTM by Word	5.60%**	4.49%**	20.26%**

Table 10: Word-level metrics comparing models on W-NUT

Model	Error Rate	Precision	Recall	F1
Preprocessing Only	9.44%	-	-	-
Term Mapping	3.04% ^{not sig}	93.33%	70.61%	80.39%
DeepNorm by Sentence	3.06%**	86.71%	71.94%	78.64%
DeepNorm by Word	2.94%*	86.95%	74.89%	80.47%
Skip-Bigram	1.67%**	90.77%	77.56%	83.64%
CLSTM by Word	9.11%*	34.99%	22.87%	27.66%

Table 11: Word-level metrics comparing models on Holmes

Model	Error Rate	Precision	Recall	F1
Preprocessing Only	22.98%	-	-	-
Term Mapping	14.27%**	80.36%	46.28%	58.73%
DeepNorm by Sentence	6.25%**	80.05%	74.11%	76.96%
DeepNorm by Word	12.37%**	68.12%	50.59%	58.06%
Skip-Bigram	11.41%**	87.86%	52.74%	65.91%
CLSTM by Word	14.05%**	62.39%	45.20%	52.42%

Table 12: Word-level metrics comparing models on Librispeech

Model	Error Rate	Precision	Recall	F1
Preprocessing Only	34.76%	-	0.00%	-
Term Mapping	37.26%**	6.51%	1.42%	2.33%
DeepNorm by Sentence	34.76%**	-	0.00%	-
DeepNorm by Word	34.76%**	-	0.00%	-
Skip-Bigram	40.04%	6.61%	2.67%	3.80%
CLSTM by Word	35.16%**	4.40%	0.23%	0.43%

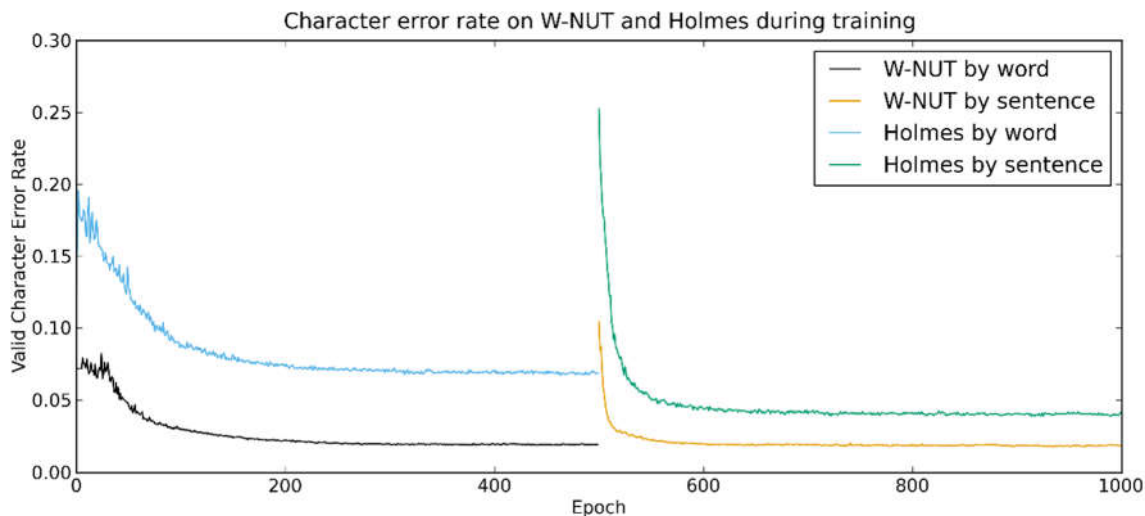


Figure 15: A comparison of the character error rate during training on W-NUT and Holmes

W-NUT

The by-word version of CLSTM was not more successful on one-to-many normalizations. In fact, it had close to 0% accuracy on these sorts of errors. We could not directly measure word accuracy on the sentence-level CLSTM, but a visual analysis of output in Table 13 suggests that it had similar trouble. Since CLSTM did not have official by-word matches, the CLSTM column was selected by hand from the predicted sentence according to its position in the surrounding words. DeepNorm by word is excluded for space.

Skip-Bigram on the other hand, designed specifically for the W-NUT dataset, has no trouble with learning frequent acronyms like “lol.” Even without an explicit vocabulary, DeepNorm is also successful in learning token-token mappings since instead of relying on a character-alignment framework it can represent each word as a whole internally to be reconstructed in output.

Working at the word level rather than at the sentence level improved DeepNorm’s performance on W-NUT. Holmes was better by-sentence, which suggests that context is more important in correcting OCR errors on Sherlock Holmes than in Twitter normalization on the W-NUT task. Holmes was split by sentence and had an average length of 15 words, the same as the average length of one of W-NUT’s tweets. The errors in W-NUT’s training set, however,

are 26% one-to-many conversions, which are largely self-contained (e.g., lol in a Tweet seldom means something other than “laughing out loud”). Table 14 shows the outputs of each model on a given tweet. Outputs of word-by-word models are concatenated with spaces as delimiters.

Table 13: Examples of one-to-many terms in W-NUT and the normalizations offered by each approach

Input	Gold	CLSTM by Word	CLSTM by Sentence	Skip-Bigram	DeepNorm	DeepNorm by Word
lol	laughing out loud	aut lou	laughing out lou	laughing out loud	laughing out loud	laughing out loud
Idk	i don't know	id kn	o tont k	i don't know	i don't know	i don't know
smh	shaking my head	smhhu	Osmh	shaking my head	shaking my head	shaking my head
Thankyou	thank you	thankyou	Thankyou	thank you	thank you	thank you
finna	going to	finna	Finna	going to	going to	going to

Table 14: An example noisy tweet and the corrections offered by each approach

Model	Example
Input	@Ez_DoesssIt yeh but still that's wild lol
Gold	@ez_doesssit yeah but still that's wild laughing out loud
CLSTM by Sentence	@ez_doesssit yeh but still thae s n laughing out lou
CLSTM by Word	@ez_doesssit yeh but still that's wild aut lou
Skip-Bigram	@ez_doesssit yeah but still that's wild laughing out loud
DeepNorm	@ez_doesssit yeah but still that's wild laughing out loud
DeepNorm by Word	@ez_doesssit yeah but still that's wild laughing out loud

Holmes

As expected, the clear winner on the Sherlock Holmes OCR dataset was CLSTM by Sentence, followed by DeepNorm and finally Skip-Bigram. Table 15 shows an example sentence and the predicted normalizations. Firstly, CLSTM by Sentence successfully corrects

all errors while adding none. Skip-Bigram succeeds in lowercasing most of the miscapitalized words, but not all of them, and is the only one of the three main models (DeepNorm, Skip-Bigram, and CLSTM by Sentence) not to correct “light.” Both versions of DeepNorm correct all mistakes, but sometimes introduce an error of their own.

Table 15: An example sentence from the Holmes OCR post-processing corpus and the corrections offered by each approach

Model	Example
Input	The light In our wIndow above showed that thIs late vIsIt was Indeed Intended for us .
Gold	The light in our window above showed that this late visit was indeed intended for us .
CLSTM by Sentence	The light in our window above showed that this late visit was indeed intended for us .
CLSTM by Word	The hight in our window above showed that this late visit was indeed intended for us .
Skip-Bigram	The light in our wIndow above showed that thIs late vIsIt was indeed intended for us .
DeepNorm	The light in our window above showed that this late visit was indeed ingerded for us .
DeepNorm by Word	The light in our window above showed that this late visht was indeed intended for us

Librispeech

Performance on Librispeech is poor for every model. DeepNorm’s approach, which came from simply learning not to try and correct anything and getting exactly baseline performance, was not significantly worse than the lowest error-rate approach by CLSTM. Technically speaking, DeepNorm’s baseline prediction occurred because of an early-stopping technique which involves saving the model weights when the character error rate is at its lowest and using these weights at test time. Predicting everything as *do not correct* is normal early behavior for the Flagger. As it ordinarily would, DeepNorm left this state after a short time in training, but since its more complex representation did not generalize from the training data to the validation data, its best fit weights, the ones used at test time, were those learned early on: the naïve approach of correcting nothing. DeepNorm’s chaotic and fruitless training on Librispeech is shown in Figure 16.

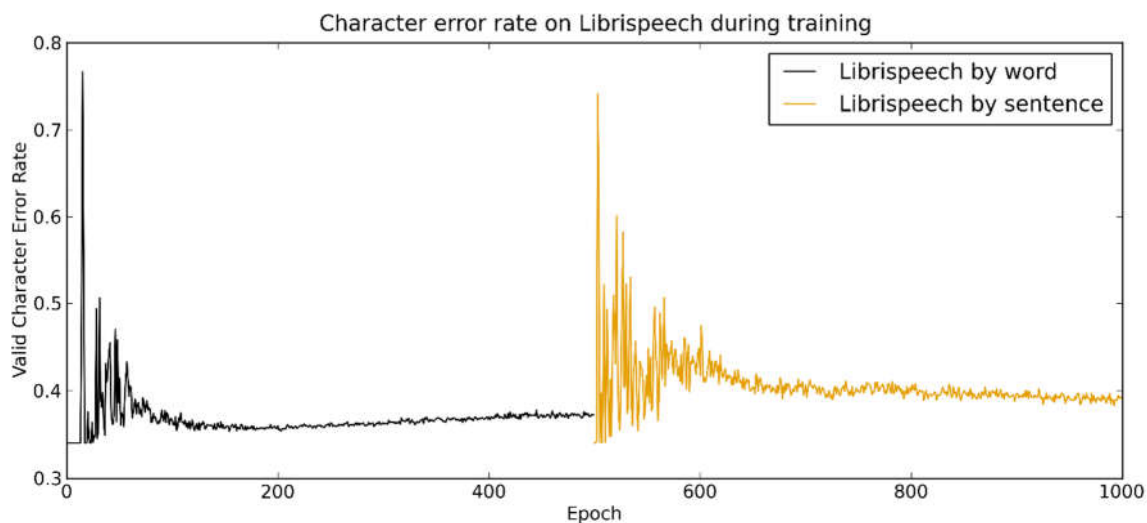


Figure 16: Character error rate on Librispeech during training

5.5 Discussion

DeepNorm’s character-word structure allows it to effectively normalize noise at both the word and character levels, permitting greater generality over competing approaches. CLSTM’s character alignment-based model, particularly effective for optical character recognition, is not effective for learning the token-token mappings common in Twitter normalization. With its overlap-based dictionary approach, Skip-Bigram is well-suited to Twitter but is less adept at learning character-level errors that persist across words. DeepNorm on the other hand can learn both word-level and character-level patterns as it takes input and constructs output character by character.

This freedom can be dangerous however, as illustrated by “ingerded” and “visht” where the flagger failed to indicate that the word did not need to be corrected and then the Reconstructor made a small mistake. We expect that unsupervised pre-training on a large example corpus of a similar domain will help to strengthen the internal language representation and reduce word misconstruction. Determining the precise manner in which to accomplish this is nontrivial. We explore one potential technique in Chapter 6.

On W-NUT both versions of CLSTM seem to fail to normalize some character-based errors such as the missing space between “thank” and “you” in Table 10. This loss may be due to a confusing effect arising from its attempts to learn word-substitution patterns its character-based model is not well-designed to represent, leading to an overall less effective model.

As we predicted it would when we introduced it in Section 5.2, the Librispeech task proved too challenging for every model set against it. Looking back to Table 8, we can see that to perform well would require a model to be able to correct such errors as “nothing can part” → “and”. We expect that these exceptionally difficult errors arise when Sphinx’s decoder, the component that translates phonemes to words, fails. As described in Section 5.2, Sphinx’s decoder uses a language model that attempts to make a reasonable sentence from the recognized phonemes. When this component fails it takes erroneous output further from its original form as well as making it more difficult to separate errors from correct text. The decoder component is standard in ASR (Jurafsky and Martin, 2008). For these reasons we suspect that for the purposes of ASR, our normalization model may be more effective replacing Sphinx’s decoder rather than operating on its output. This would be an interesting area for future work.

In the next chapter we run several studies examining DeepNorm and continue this discussion in more detail.

CHAPTER 6

ANCILLARY EXPERIMENTS

In this chapter we report on several additional experiments to explore questions related to our model and Lexical Normalization in general. These experiments include ablation studies to explore the design of DeepNorm from Chapter 6, an investigation of unsupervised pretraining to support DeepNorm, and a brief exploration of lexical normalization’s potential for improving performance of downstream tasks. Because no model performed well on Librispeech, we exclude it in most of these studies.

Given that training without context is a necessary pretraining step to the context-based model, it is straightforward to include context and non-context versions of DeepNorm in most of the studies below. We discuss the effects of context on each of these studies, and show the effects of context on the full model by looking at the full model results included in each study for comparison.

6.1 Cox Windowing

To examine the benefits of Cox Windowing, as described in Chapter 5, we evaluate our work on the W-NUT and Holmes datasets without it. When we do not use Cox windowing, what we are using is the padded window approach described in Chapter 3. The results are shown in Table 16 and Table 17. As in previous chapters, we use approximate randomization ($R=10,000$) for significance. A single asterisk indicates $p<0.05$, and double indicates $p<0.001$. All the graphs in this chapter are scaled to best fit the contained plots.

We see in our results that Cox windowing does help performance. The effect is clear in Holmes. Although it is more subtle in W-NUT, it is in most cases still significant. The effect on character error rate is strongest by word, but on word error rate is strongest by sentence. We suspect that by word, the model benefits more from a more sophisticated character level analysis because it is not diluted by the addition of context-level modeling. With the addition of context-level modeling, the extra sophistication of character level analysis may disproportionately help avoid “off by one” errors in which a word is wrong by only one character and therefore that contribute more to word error than to character. These errors can

become more prevalent in the presence of noise added by DeepNorm’s context modeling component.

Figure 17 and Figure 18 show the learning pattern when removing Cox windowing. Probably the most interesting pattern is that which arises in Holmes, where the by-word training is much more chaotic at first without the benefit of Cox windowing than with. The most straightforward interpretation is that this is a direct reflection of the stability offered by giving suffixes as well as prefixes guaranteed positions in a window.

Table 16: Results on W-NUT with and without Cox windowing

Windowing	Context	Character Error Rate	Word Error Rate	Precision	Recall	F1
Padded	DeepNorm by Word	1.43%	3.04%	86.14%	73.67%	79.42%
	DeepNorm by Sentence	1.45%	3.26%	82.49%	71.25%	76.46%
Cox	DeepNorm by Word	1.36%**	2.94%*	86.95%	74.89%	80.47%
	DeepNorm by Sentence	1.45%	3.06%**	86.71%	71.94%	78.64%

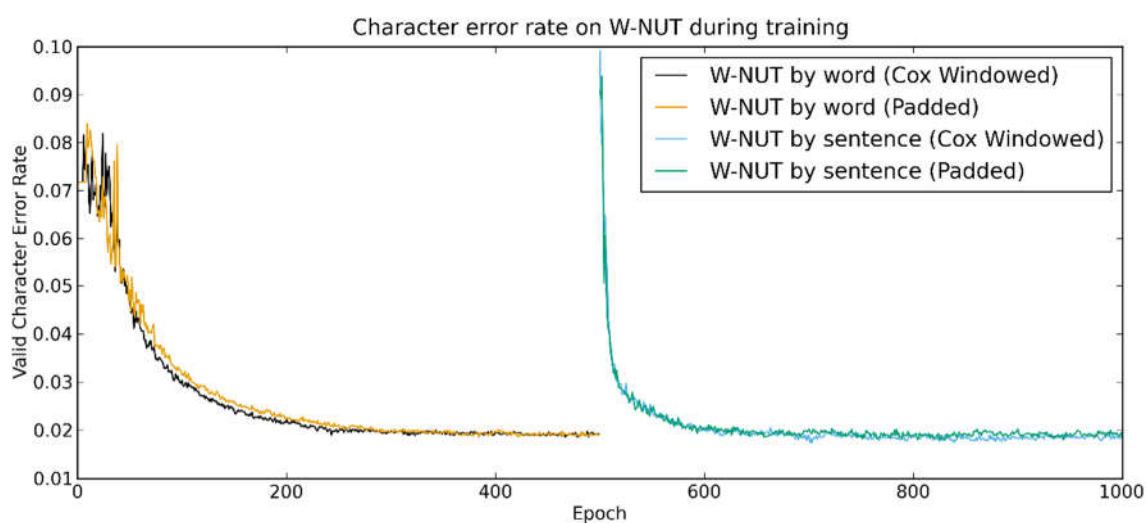


Figure 17: Effects of removing Cox windowing on training of W-NUT

Table 17: Results on Holmes with and without Cox windowing

Window	Context	Character Error Rate	Word Error Rate	Precision	Recall	F1
Padded	DeepNorm by Word	6.73%	15.80%	61.00%	39.26%	47.77%
	DeepNorm by Sentence	4.49%	9.71%	73.47%	62.37%	67.46%
Cox	DeepNorm by Word	4.64%**	12.37%**	68.12%	50.59%	58.06%
	DeepNorm by Sentence	2.69%**	6.25%**	80.05%	74.11%	76.96%

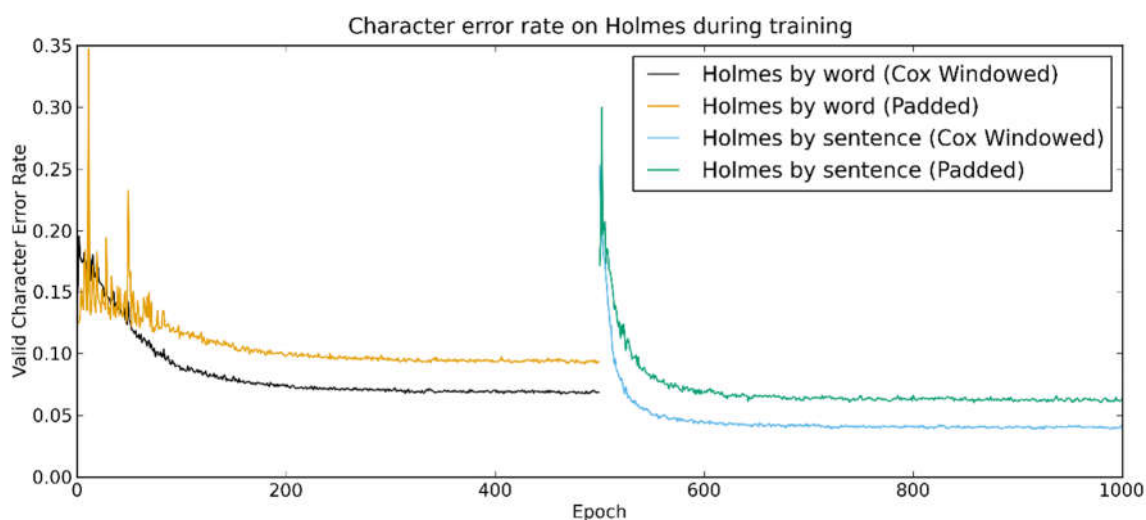


Figure 18: Effects of removing Cox windowing on training of Holmes

6.2 Tweetcode Exclusion

Exclusion of tweetcode (@at-mention, #hashtag, http://url.com) in training on W-NUT exists in an unusual space between a task-specific feature and a generally applicable rule. While it falls comfortably within the realm of selecting from datasets rather than adding hand-engineered features in our work, it nevertheless is the most task-specific part of this study. We used it in the official W-NUT competition described in Chapter 3, but not in our later work

described in Chapter 5. We excluded it at this point for the purposes of making a system whose absence of task specific features was as unambiguous as possible. In this section we examine the effects that exclusion of tweetcode had on performance. Our results are shown in Table 18 and Figure 19.

Unexpectedly, we found that excluding tweetcode, when it had any significant effect at all, harmed rather than helped performance. Figure 19 shows that, by word, tweetcode exclusion helps its model get an early lead, but eventually including tweetcode wins out. To consider why the exclusion of tweetcode may have done more harm than good, we must delve into the implementation of the exclusion feature. Selective exclusion of tokens from consideration when training a normalizer is more difficult than it may at first seem, especially when working with context. First of all, although we exclude tweetcode from normalization, we still must include it as context. So, our excluder does not simply prevent input from being seen. Instead, the model does compute the tweetcode input. Once it has calculated word vectors and included them as the context informing the words that are to be normalized, it can then exclude them when calculating cost. In addition, while we exclude tweetcode from consideration when training the Reconstructor, we must include all data when training the Flagger. While this was straightforward in Chapter 3 when the Reconstructor and Flagger were separate models, it became less so when we combined them into one model.

What we did to train the Flagger of our unified model separately on the excluded words was to keep track of the words we had excluded from the full training and calculate their cost separately based only on the Flagger. Our mistake was to then add that cost to the previous cost. As both costs were averages, the model may have been weighted unnecessarily towards learning how to exclude tweetcode at the expense of other phenomena, such as reconstruction, that should be taking more of its attention. That is, since there was less tweetcode than there was of other words, it should have contributed less to the cost, but by averaging each separately and adding them together, it contributed equally. This could have been further exacerbated with the greater complexity and information availability of the context-based model, which would have greater opportunity to overfit to noise based on the misweighting. This should

serve as a valuable lesson about the challenges of building one neural network to make multiple separate decisions.

Table 18: Effects of Tweetcode exclusion on results

Tweetcode	Context	W-NUT				
		Character Error Rate	Word Error Rate	Precision	Recall	F1
Exclude	DeepNorm by Word	1.45%	3.1%	85.43%	73.92%	79.26%
	DeepNorm by Sentence	1.43% ^{not sig}	3.1%	85.83%	72.23%	78.44%
Include	DeepNorm by Word	1.36%**	2.94%**	86.95%	74.89%	80.47%
	DeepNorm by Sentence	1.45%	3.06% ^{not sig}	86.71%	71.94%	78.64%

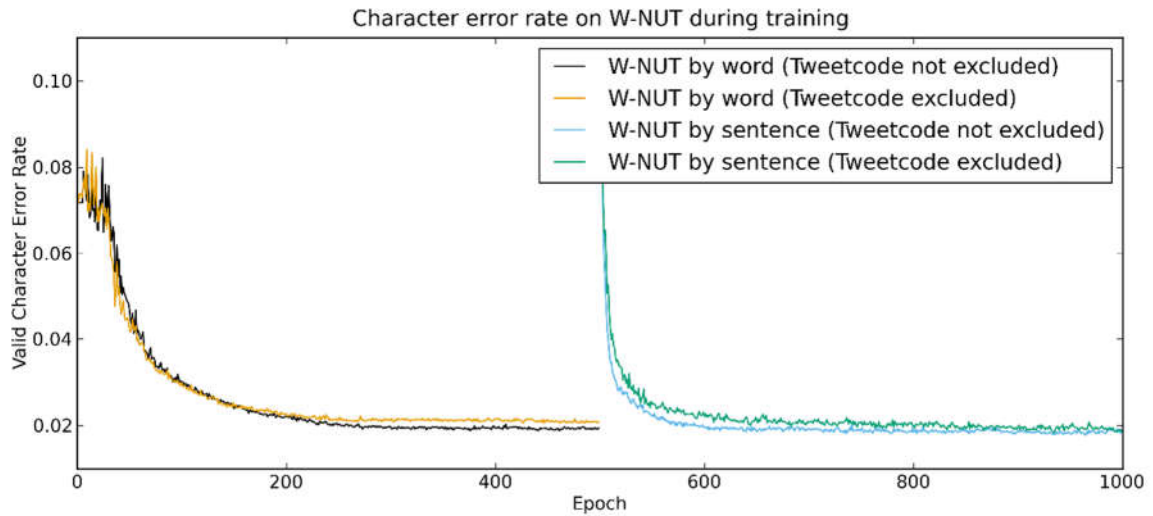


Figure 19: Effects of Tweetcode exclusion on training

6.3 Unified Reconstruction and Flagging

In Chapter 3, we describe the Flagger and the benefit it provides to our normalizer. Here the Flagger is a separate model from the normalizer. Each is trained separately and has hyperparameters optimized separately. In Chapter 5, we cut the parameters that we need to learn in half by unifying these models. Having one model also leads to better theoretical elegance and gives the opportunity for superior generalization as the model is forced to learn features at lower levels that apply to both the Flagger and the Reconstructor.

We have already discussed some of the issues with attempting to unify reconstruction and flagging and creating a single model that makes multiple different predictions. In Section 6.2, we described the challenge of excluding certain irrelevant data that might harm the effectiveness of the Reconstructor without excluding it from the Flagger, where it is essential. Another element that may weaken the unified model is the inability to specialize hyperparameters for different predictions. In Chapter 3 we found that best validation performance occurred for the Flagger with many fewer parameters and much higher dropout than it did with the Reconstructor. This matched what one might expect for the relatively simple task of determining whether to correct a word compared to the more complex task of actually reconstructing words to their normalized forms character by character. DeepNorm’s unified model does not support this specialization as only one set of hyperparameters must apply to both the Reconstructor and the Flagger.

To evaluate the effect of unification on DeepNorm, in this section we run a study in which we train a DeepNorm Flagger and a DeepNorm Reconstructor model separately for each of W-NUT and Holmes. We show the learning pattern of the Flagger and the Reconstructor in each case and then we compare results to our canonical unified DeepNorm model. Our results are in Table 19 and Table 20. Figure 20 and Figure 21 show the learning pattern of the separate Flagger and Reconstructor. Notice how quickly the Flagger converges compared to the Reconstructor.

We see that both of our models perform significantly better when unified. This is an especially interesting result because on W-NUT our unified model performs slightly worse than the separate model from Chapter 3. A likely explanation is that for the purposes of this

simple test we did not take the time to optimize hyperparameters separately for the Flagger and the Reconstructor, and instead left the hyperparameters as they were. So, what we are seeing is each model overfitting as its parameters stay the same while the complexity of its task is cut roughly in half.

An especially interesting phenomenon here is the reversal of the effect seen in the Cox windowing study. Rather than having a muted effect, the change has the stronger effect on W-NUT than on Holmes. We suspect that the reason is in fact the same. As a simpler task, W-NUT gets less advantage from extra information provided by Cox windowing, but is hurt more by excess model complexity allowing for overfitting to the relatively simple data.

Table 19: Separate output results versus unified on W-NUT

	Model	Character Error Rate	Word Error Rate	Precision	Recall	F1
Separate	DeepNorm by Word	1.53%	3.34%	83.80%	69.70%	76.11%
	DeepNorm by Sentence	1.86%	3.85%	81.78%	62.25%	70.69%
Unified	DeepNorm by Word	1.39%**	3%**	86.94%	74.35%	80.16%
	DeepNorm by Sentence	1.45%*	3.06%**	86.71%	71.94%	78.64%

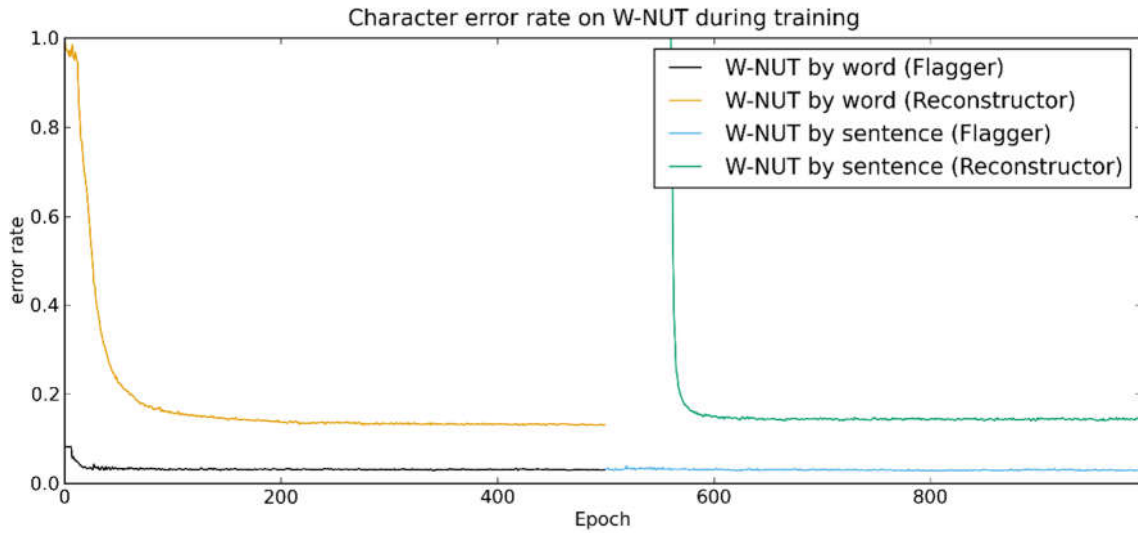


Figure 20: Flagger and Reconstructor training on W-NUT

Table 20: Separate output results versus unified on Holmes

	Model	Character Error Rate	Word Error Rate	Precision	Recall	F1
Separate	DeepNorm by Word	4.80%	12.54%	67.18%	49.50%	57.00%
	DeepNorm by Sentence	2.99%	6.95%	76.99%	71.32%	74.04%
Unified	DeepNorm by Word	4.64%**	12.37%**	68.12%	50.59%	58.06%
	DeepNorm by Sentence	2.69%**	6.25%**	80.05%	74.11%	76.96%

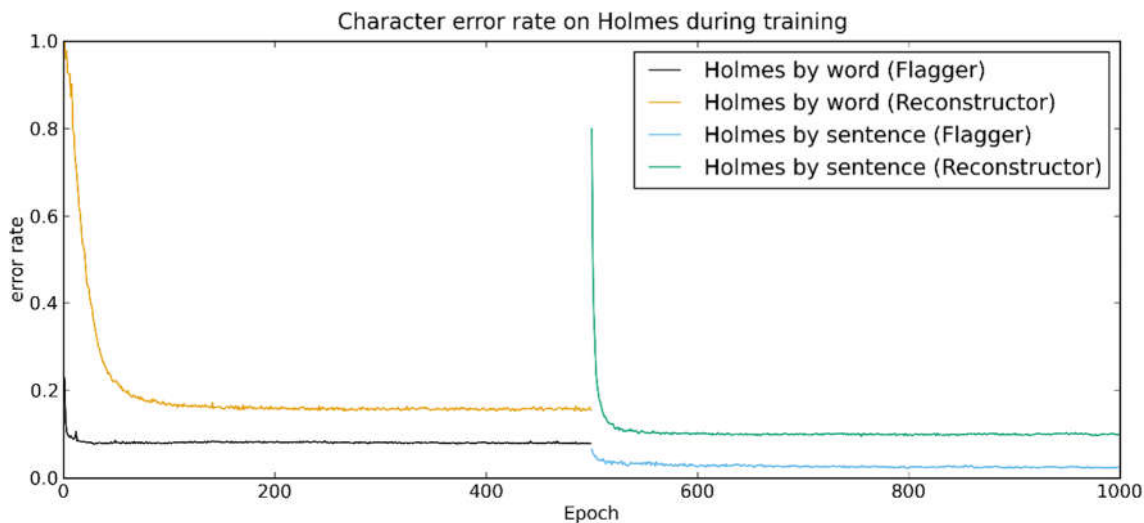


Figure 21: Flagger and Reconstructor training on Holmes

6.4 Unsupervised Pretraining

Although the benefits of being able to apply a single architecture to multiple domains of lexical normalization without the need for labor-intensive feature engineering should not be understated, the benefits of applying a single model, that is, one that does not need even to be retrained for each domain, are dramatically greater. Nevertheless, the challenge of such a model is also far greater, so incremental research towards this goal takes many forms. A first step is to develop techniques to use unannotated data to train a model.

It is an accepted phenomenon in machine learning in general, no less deep learning, that more data is almost always better for task performance. The data annotation necessary for supervised learning, although it can often be accomplished by less skilled workers than feature engineering, still tends to be a costly and labor-intensive pursuit. The advantage of being able to learn from data without human annotation is apparent. We refer to this as *unsupervised learning*. Learning from unsupervised data brings with it its own set of challenges, however, so combining unsupervised and supervised approaches is another strategy. The strategy we will use in this paper uses unsupervised learning to initialize weights in preparation for supervised learning, *unsupervised pretraining*.

Unsupervised Learning Methods

In this work we explore the use of synthetic noising to make noisy-normal pairs out of clean data and how this type of synthetically noised data can be used for unsupervised training of our normalization model. Although the idea of making more noisy-normal pairs suggests supervised rather than unsupervised learning, this process of applying noise to an input and reconstructing the original as a means of learning without the benefit of annotation has precedent in the training of denoising autoencoders (Vincent et al., 2008). Our noising technique is to use a simple character substitution model. For each letter in a word, we have a fixed probability p that another character from the model’s alphabet will be substituted for that character. We deliberately do not select a noising technique in order to match a particular noise style, such as optical character recognition noise. What we are attempting to accomplish here is to build a morphological and contextual language model that will be able to inform the supervised model that we then train based on it.

We selected 0.3 for our p value. Approximately a third of letters would be randomized, leading to a challenging task, but not an impossible one. We select 0.3 as a simple value between 0.2 and 0.5, two levels of noise used in work on image analysis (Srivastava et al., 2014; Vincent et al., 2008). In this work noise was added by dropping values out, that is, coercing them to zero rather than substituting random values. Our approach makes it more difficult to tell a wrong word from a right word, which is one of the most important aspects of text normalization. A given image also has more pixels than a given word has letters. Therefore, it is of more detriment to lose a third of the letters in a word than to lose a third of the pixels on image, as in many media, images and text included, a lost positional input can be approximated from its nearest neighbors, and a lost letter will have fewer adjacent neighbors than a lost pixel. It is for these reasons that we choose a noising rate closer to the smaller value.

Unsupervised Data for W-NUT

As a source for our unsupervised data we use the National Institute of Standards and Technology’s 2011 Twitter dataset (Han and Baldwin, 2011). This data set consists of 16 million tweets, from which we extracted approximately 8 million English tweets using the

python program *langid* (Lui and Baldwin, 2012). One issue specific to our work is that the data naturally available for Twitter is in its noisy form whereas the data naturally available for OCR and ASR are in their normal form. To handle this issue, we take advantage of the fact that much of the text on Twitter is already normal, and use an assumption with prior representation in the literature that a tweet with no out-of-dictionary words aside from tweetcode is a normal tweet (Han and Baldwin, 2011). In this manner we extracted approximately a third of tweets from our English tweets dataset to make a dataset of 17,000,000 words we refer to as *Clean NIST*. A selection of Clean NIST is shown in Table 21. Tweets are reconstructed from their tokenized words using space as the delimiter. While the clean tweets still include slang, they represent the normal text found on Twitter better than either the noisy tweets or text from another source. Remember that all output is lowercased in the W-NUT task and that in lexical normalization grammatical issues are ignored.

Unsupervised Data for Holmes

The Holmes unsupervised data comes from the remainder of the Adventures of Sherlock Holmes that was not used in making the Holmes supervised dataset. This constitutes roughly 500,000 words. As sentences were selected randomly from the series of short stories to make the supervised dataset, the unsupervised data is necessarily selected randomly as well. Even so, being the exclusive product of a single author, this is both the smallest and most domain-specific unsupervised dataset of the three that we use. Despite the size, we expect the domain specificity to make it more effective for the task.

Unsupervised Data for Librispeech

We use the train-clean-100 dataset for our unsupervised Librispeech data. It consists of 100 hours of what the creators considered easily understood readings of classic texts. Because we were only in need of unsupervised text, we took the transcripts directly and did not use the audio. This resulted in roughly 4,500,000 words of text from classic English-language literature.

We note here that with these large data sets, we split the data into sets of approximately 250,000 words each. Epochs are calculated according to the number of 250,000 word subsets

iterated over rather than iterations over the complete dataset. We choose 250,000 words as the largest number of words that would reliably fit on the 5 GB of memory on our GPU while leaving space for active computation. Because transferring data to and from GPU memory is expensive relative to performing computations on a GPU, it is our goal to minimize the amount of necessary data stored outside the GPU.

Table 21: Randomly Selected Tweets from Nist’s noisy and clean datasets

Clean Tweets	Noisy Tweets
Chef salad is calling my name , I'm so hungry !	@KissMyTweetMuah word thats how u feel lol
@Chalk_Flew_Up Yup . Maybe not for much longer .	Meeeeeee ! RT @missLOVElace_ : who wants my 80,000 tweet ?
#ZodiacFacts #Sagittarius have enormous confidence in themselves . They feel like if you don't have confidence then you don't have nothing .	#np Preacher Man - Asa ... #GreatMusic
@BeasBookNook every ARC I've ever read hasn't had final copy edits . Isn't that the nature of the beast ? I guess some are better than others .	@MariNashia n____n de meexicoow y tu
Instead of watching Harry Potter in french maybe i should be getting ready ...	Check our FB promo to #win #free stuff : http://wfi.re/5h4zm Grand Prize = #ski package to any resort in Nth America ? #ski #snowboarding

Results

Our result graphs are given in terms of validation data on the final task, even during unsupervised pre-training. Training on these gigantic corpora was much slower than training on the small supervised datasets, so we halted training at each stage when it no longer appeared to be improving. Looking at Figure 22 and Figure 23, we see a trend. Performance seems to converge near the baseline for the by-word model, but by-sentence quickly diverges. We expect that this is due to overfitting on the unsupervised data. This is not as much a problem as it is in supervised learning because there is still another supervised step after unsupervised

pre-training that will fix anything that has been overfit to the unsupervised data. Figure 22 also shows that training unsupervised on Librispeech is just as chaotic as training supervised.

After unsupervised pre-training, supervised training looks much like training from random initialization. We have three different training paradigms that we consider. We list the training orders below:

- Unsupervised by word → Supervised by word
- Unsupervised by word → Supervised by word → Supervised by sentence
- Unsupervised by word → Unsupervised by sentence → Supervised by sentence

We show results of unsupervised both alone and as a pretraining step in Table 22, Table 23, and Table 24. We compare the best model in each dataset to the corresponding supervised-only version listed in Chapter 5 for significance.

As we expected, our deliberately generic and simple unsupervised training alone did not result in worthwhile models. We found that skipping supervised by word was bad for performance. Because our unsupervised approach is focused on modeling the expected forms of words in normal text and uses a generic source of noise, this result suggests that pre-training using a by-word model is important for learning the noise model in addition to the standard forms of words.

Pretraining unsupervised by word and then training supervised by word and supervised by sentence, essentially taking our original approach with the additional unsupervised pretraining by word step, proved to be the best model for Holmes. This model proved to be significantly better than the fully supervised version and extremely significantly better with regards to word error rate. W-NUT's approach of unsupervised by word pretraining and then supervised by word training is superior to the fully supervised approach, but not significantly so.

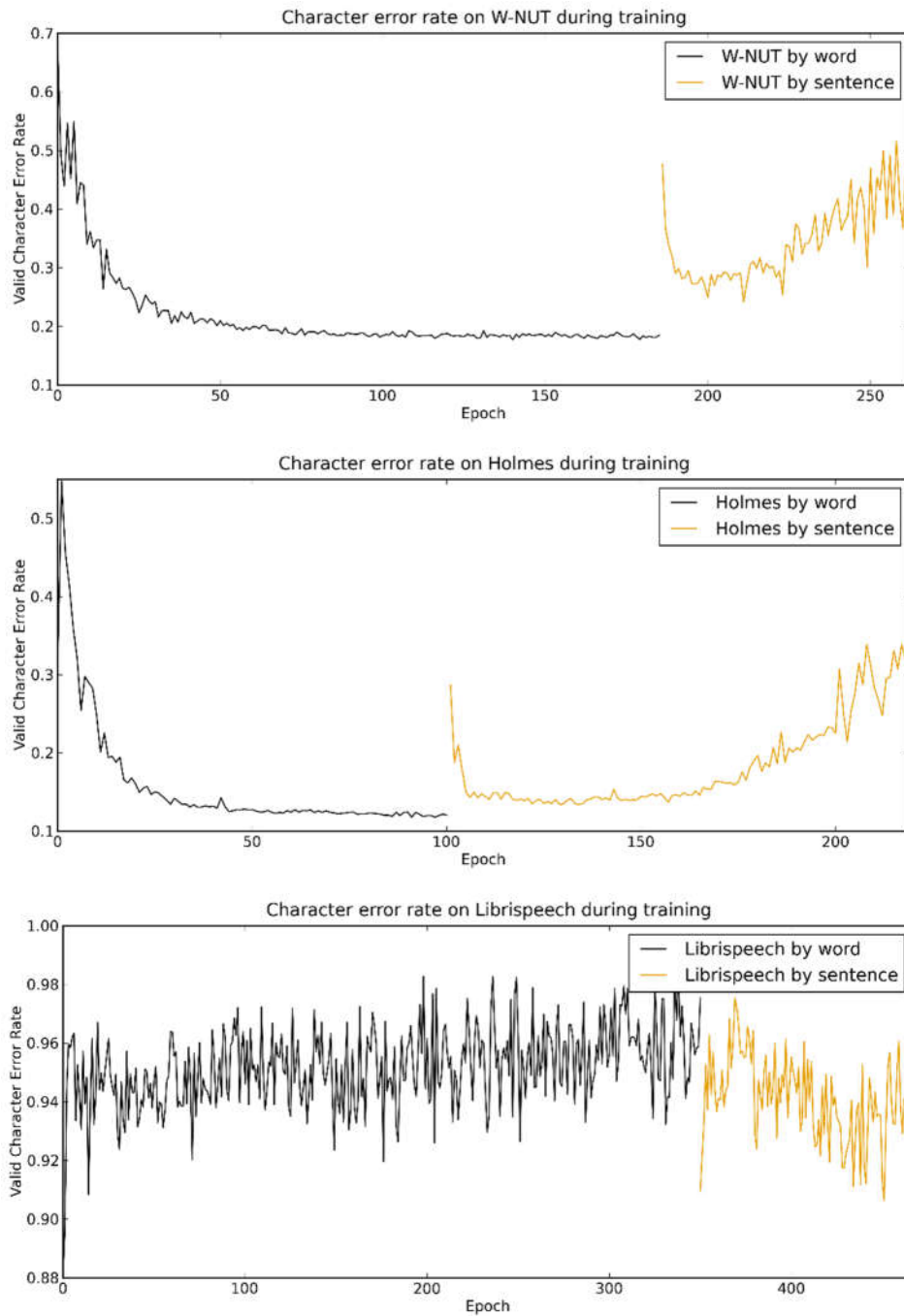


Figure 22: Character error rate on supervised task during unsupervised pre-training

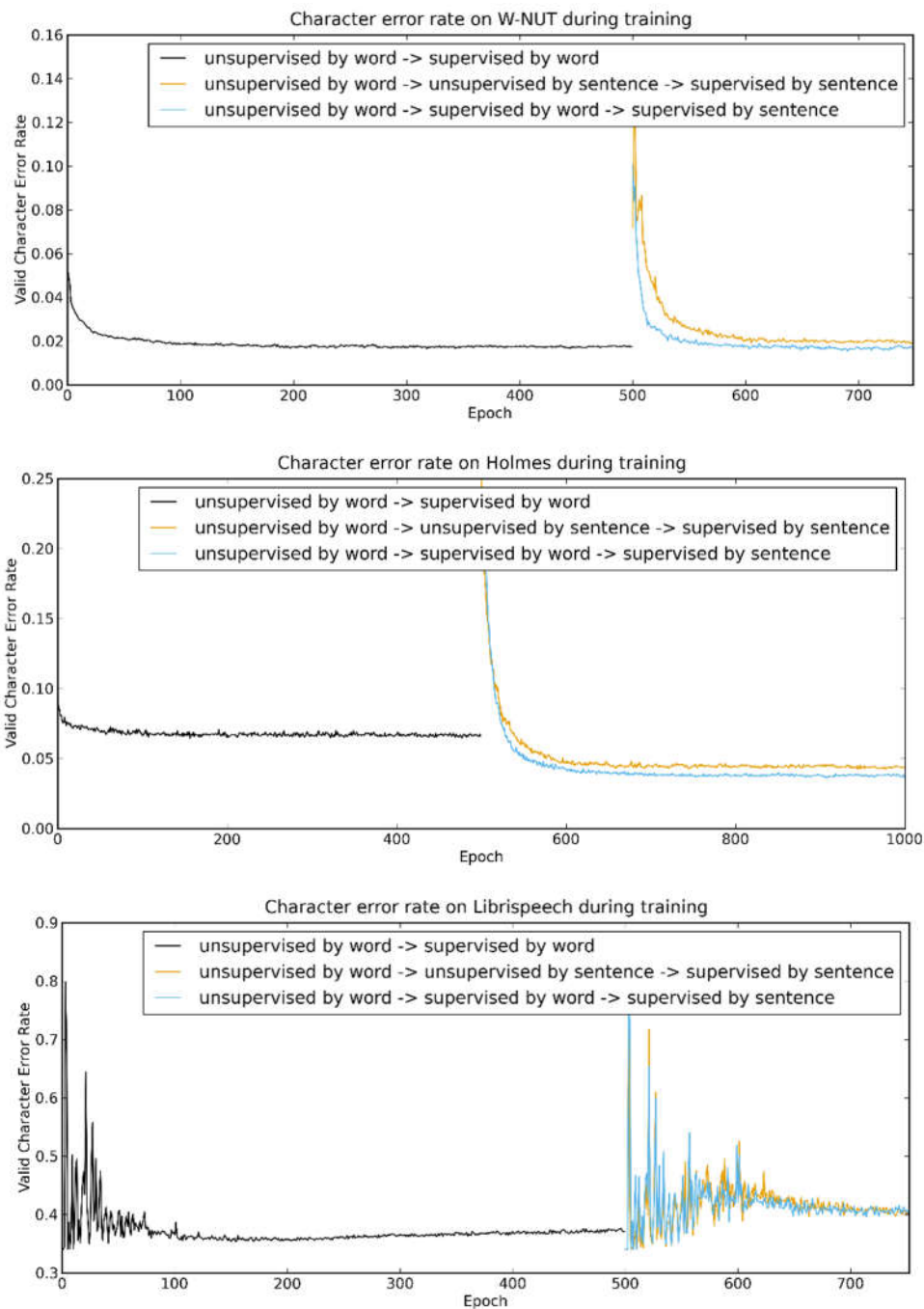


Figure 23: Performance during supervised training after unsupervised pre-training

Table 22: Unsupervised pretraining results on W-NUT

Model	Character Error Rate	Word Error Rate	Precision	Recall	F1
Unsupervised Word	15.61%	23.98%	0.47%	0.83%	0.60%
U Word \rightarrow U Sentence	20.60%	24.06%	0.39%	0.72%	0.51%
U Word \rightarrow S Word	1.34% ^{not sig}	2.93% ^{not sig}	86.87%	74.86%	80.42%
U Word \rightarrow S Word \rightarrow S Sentence	1.38%	3.11%	84.16%	73.09%	78.23%
U Word \rightarrow U Sentence \rightarrow S Sentence	1.61%	3.41%	81.99%	69.70%	75.35%

Table 23: Unsupervised pretraining results on Holmes

Model	Character Error Rate	Word Error Rate	Precision	Recall	F1
Unsupervised Word	8.90%	23.76%	10.56%	5.37%	7.12%
U Word \rightarrow U Sentence	10.30%	24.88%	8.88%	6.26%	7.34%
U Word \rightarrow S Word	4.53%	12.37%	68.03%	51.00%	58.30%
U Word \rightarrow S Word \rightarrow S Sentence	2.63%*	6.05%**	79.91%	75.11%	77.43%
U Word \rightarrow U Sentence \rightarrow S Sentence	3.17%	6.93%	75.76%	71.11%	73.36%

Table 24: Unsupervised pretraining results on Librispeech

Model	Character Error Rate	Word Error Rate	Precision	Recall	F1
Unsupervised Word	69.85%	98.87%	0.01%	0.03%	0.02%
U Word \rightarrow U Sentence	68.32%	94.70%	0.34%	0.92%	0.5%
U Word \rightarrow S Word	20.25%	34.76%	-	-	-
U Word \rightarrow S Word \rightarrow S Sentence	20.25%	34.76%	-	-	-
U Word \rightarrow U Sentence \rightarrow S Sentence	20.25%	34.76%	-	-	-

6.5 Downstream Analysis

As described in Chapter 1, the primary benefit of text normalization is to provide improvement to downstream tasks. In order to begin the process of learning how much benefit we can gain from normalization, we performed a simple test on Twitter data normalized by our approach in Chapter 3. First, we used the Stanford Part of Speech tagger (Toutanova et al., 2003) on a

Twitter data part of speech tagging task (Owoputi et al., 2012). We do not use a Twitter Part of Speech tagger because the goal of normalization is to improve results on systems not designed for noisy text. The task was a supervised one so we trained the Stanford tagger on the training data and tested it on the test data. Then we applied our normalizer to both test and training data and trained and tested the Stanford tagger again. Our results in Table 25 show a small improvement from our normalizer. This is encouraging, and we expect better normalization on more complex tasks to provide more improvement, and especially on tasks for which the training data and test data are not noised in the same way.

Table 25: Statistics on Stanford Part of Speech on normalized and raw Twitter data

Data	Sentence accuracy	Tag accuracy	Unknown Words	Unknown Word accuracy
Not Normalized	20.9%	84.0%	8060	64.6%
Normalized	21.1%	84.3%	8046	65.0%

6.6 Discussion

In this chapter we explored the effects of making various alterations to our architecture. Contrary to expectations, several of the elements we explored were not of unambiguous benefit to our model. Cox Windowing is an exception, providing significant improvement over a pure padded approach in all but one case, and having no significant effect in that case. Unification appears to help on both W-NUT and Holmes, but since it did not adjust hyper parameters, our study was not conclusive. Other changes are contrary to expectation. Exclusion of tweetcode when training the Reconstructor is harmful rather than helpful to performance, which may be the result of an implementation issue rather than the concept itself. Context is important for Holmes but harmful for W-NUT. Similarly, unsupervised pre-training is helpful for Holmes but has no significant effect on W-NUT.

The difference between W-NUT and Holmes in each experiment is particularly interesting. The effect of losing Cox windowing appears much more profound on the Holmes

datasets than the W-NUT dataset, context hurts W-NUT rather than helps, separation without adjusting hyper parameters is vastly more harmful on W-NUT than on Holmes, and unsupervised pre-training does not help W-NUT. All of these phenomena can be explained at least partially by W-NUT's relative simplicity as a task. W-NUT's errors are 26% one-to-many conversions, mostly acronym expansions that with a few variations are relatively easily mapped from one to another. This may give it a high floor, after which it is more difficult to make significant gains even with a more sophisticated model. This would explain the architectural changes that lead to relatively small or insignificant gains such as unsupervised pre-training and Cox windowing.

It may also be easy to fall below this floor with a more complicated model that overfits to irrelevant noise. This would explain the significant rise in error rate when effectively doubling the parameters by using two models instead of one and the addition of context analysis, especially since acronym expansions are largely unaffected by context. The fact that W-NUT is deliberately simplified by lower-casing output and that it does not involve any punctuation errors, unlike the missing periods in Holmes, may also lead to it being a relatively simple task prone to overfitting rather than benefiting from powerful representations.

Finally, another explanation for the minimal improvement on W-NUT from the addition of unsupervised pre-training in particular may come from the fact that many corrections, such as expansions of emotive acronyms, were not ones that would appear in clean tweets. People who write tweets without nonstandard words do not tend to write "laughing out loud" as a placeholder for the common lol. This is a good example of what could be described as a weakness of W-NUT's chosen standard form. This is less a judgement of W-NUT as a dataset as a demonstration of the challenges of dealing with the absence of an unambiguous standard form for unedited text, as described in Chapter 1.

CHAPTER 7

CONCLUSION

This dissertation addresses the research question:

To what degree can the various tasks involving correcting noisy words into a normal form be combined under the umbrella task of lexical normalization, and how well can a single deep learning architecture address them as different instantiations of this generalized task?

As seen in prior chapters, we developed a technique based on deep feed-forward neural networks to normalize Twitter text without using hand-engineered features specialized to a particular type of normalization. Following the success of this architecture, we then collected and created three datasets in order to evaluate our architecture’s ability to generalize between different lexical normalization tasks. Using bidirectional gated recurrent neural networks, we expanded our approach to include surrounding words and our single approach proved comparable to specialized approaches designed for each of the optical character recognition and unedited text tasks. Furthermore, our approach outperformed the specialized approaches on the tasks for which these specialized approaches were not designed. This constitutes a significant contribution to the research literature, but there is much yet to be examined and interpreted.

7.1 Hypotheses Revisited

- H1_a. A single end-to-end architecture based on neural techniques can perform a supervised lexical normalization task on unedited text competitively with a state of the art skip-bigram model.
- H1_b. A single end-to-end architecture based on neural techniques can perform a supervised lexical normalization task on unedited text competitively with an off-the-shelf optical character recognition post-processing model.

The results show that DeepNorm performs competitively with the skip-bigram model on the W-NUT dataset of unedited Twitter text and significantly better than the OCR system CLSTM. DeepNorm performs better without context on W-NUT, perhaps because of the relative simplicity of the task and the potential of overfitting to a small dataset. Exclusion of tweetcode turns out to hurt rather than help, which could possibly be remedied by better implementation of exclusion. Cox-Windowing is significantly helpful, but only to a small extent. Overall, alterations to DeepNorm have much less effect on performance on W-NUT than they do on the Holmes dataset, suggesting that this task may be simpler than it appears.

H2_a. The architecture from Hypothesis 1 will perform competitively with the skip-bigram model on the supervised task of normalizing optical character recognition output.

H2_b. The architecture from Hypothesis 1 will perform competitively with the optical character recognition post-processing model on the supervised task of normalizing optical character recognition output.

DeepNorm outperforms the skip-bigram model significantly on the Holmes dataset and performs comparably to CLSTM, with context sensitivity and Cox-windowing being critical elements. The inclusion of capitalization and the fact that Tesseract would sometimes drop punctuation likely contribute to the importance of context sensitivity, as the position of a word in a sentence is highly relevant to whether it should be capitalized and whether a period should appear after.

H3_a. The architecture from Hypothesis 1 will perform competitively with the skip-bigram model on the supervised task of normalizing automatic speech recognition output.

H3_b. The architecture from Hypothesis 1 will perform competitively with the optical character recognition post-processing model on the supervised task of normalizing automatic speech recognition output.

Librispeech, because it alternated between completely correct and completely indecipherable text, turned out to be too difficult for any model. Thus the models were comparable by default. Librispeech’s learning pattern is also distinct from those of other models, with a vastly larger degree of jitter that illustrates the difficulty of the task.

As secondary work, we also explore the potential for learning from unannotated data and from data annotated for different normalization tasks. Our secondary hypotheses are thus as follows:

H4: There exists a combination of annotated unedited text, annotated text from other tasks, and unannotated text such that the architecture from Hypothesis 1, trained on this data and evaluated on unedited text, will achieve significantly higher performance than the same model trained using only the evaluation task’s annotated text.

Our attempt at pre-training using reconstruction of synthetically randomized data proved insufficient to significantly improve our architecture’s performance on Twitter, demonstrating the challenge of unsupervised pretraining for lexical normalization. The nature of the normalization standard may be relevant to this result. The expansion of emotive acronyms, for example, is not something that can be learned from clean or noisy data, given that “laughing out loud” is not generally substituted for “lol” in naturally occurring text on Twitter or elsewhere.

H5: There exists a combination of annotated unedited text, annotated text from other tasks, and unannotated text such that the architecture from Hypothesis 1, trained on this data and evaluated on optical character recognition text, will achieve significantly higher performance than the same model trained using only the evaluation task’s annotated text.

The Holmes dataset benefited the most from its unsupervised pre-training, achieving significant improvement over its fully supervised equivalent. Although it had the smallest pre-

training, corpus, it was also from the same specific domain, and the noising process, operating at the character level, was incidentally more like the real noise present in Holmes than in W-NUT or Librispeech. Even with these advantages, however, we see in Holmes that unsupervised pretraining has promise for helping supervised text normalization tasks.

H6: There exists a combination of annotated unedited text, annotated text from other tasks, and unannotated text such that the architecture from Hypothesis 1, trained on this data and evaluated on automatic speech recognition text, will achieve significantly higher performance than the same model trained using only the evaluation task’s annotated text.

The Librispeech dataset was too difficult for any task, and it did not have more success after unsupervised pretraining. Both the chaotic appearance of performance graphs in learning and the results in which the baseline of correcting nothing is better than many of the competing approaches indicate that the self-correction inherent in CMU-Sphinx that takes a wrong analysis further from its original inputs may make resulting outputs uninterpretable for post-processing.

7.2 Limitations

In this work we developed an architecture that can learn to normalize noisy text from different sources. One of the biggest limitations of our work is that it was only evaluated on two corpora, not counting Librispeech. To better support our claim that it can be applied to multiple types of noise, we would want to have data representing more types of noise. We also would benefit from evaluation on noisy text of additional languages. Another limitation is that although our approach successfully generalizes as hypothesized, the chosen architecture has not been extensively compared to other variations. It's likely that with more development, our architecture has considerable room for improvement. In addition, the fact that our architecture relies on supervised learning using annotated data is a limiting factor with regards to its usefulness in practice, so it is worthwhile to pursue means of making it less reliant on hand-annotated data. A fourth limitation is of normalization as a concept. As we described in the introductory chapter, normalization requires a standard to which to normalize, and the mapping

between this standard and the observed text is often challenging for human normalizers to agree upon.

7.3 Future Work

Developing an approach to text normalization that does not require feature engineering opens the door to a collection of other opportunities. As a result of our work, we are able to develop normalizers for new domains using only examples of noisy data matched to their hand normalizations, something that can be collected by people with a minimum of prior skill and training. The next opportunity to further streamline this process will be to learn from unannotated data, followed by an understanding of the relationship between noisy and normal text in general. For instance, an advanced machine learning algorithm in the future may be able to, like an educated human, read text with a completely unseen type of noise, such as “The Phaomneil Pweor of the Hmuan Mnid” with no need for examples from similarly noised text.

Even without these tremendous advances, DeepNorm could be used to continually normalize information from Twitter and other such social media as it is posted. It could automatically solicit help from a community of volunteers or a small paid staff to learn words it did not understand. This tool could provide businesses and other organizations with easily analyzable translations of the latest developments in social media culture.

Finally, although our approach is designed to output normal text, if we remove the last layer, it is simply a means of generating a vector that describes a word. This means that we can use our approach to generate word vectors. Any approach that can use word vectors can take advantage of our approach’s ability to analyze noisy text. This is especially true of deep learning approaches that could be placed on top of our approach such that DeepNorm and it could learn their weights simultaneously. Because the set of existing natural language processing techniques still relies directly on words rather than word vectors, this approach will require more changes to these approaches than a normalization technique. However, for the sentences such as “my little boi iz growin up soooo fast <3 <3 <3,” with multiple layers of meaning, direct vector representation may ultimately be the key to analyzing noisy text without semantic or pragmatic loss.

7.4 Summary

A key challenge posed by much naturally text is the tendency for it not to conform to a general standard form, making it difficult to apply approaches built for one form of text to another. The closest form to a universal standard form, the *normal* form, is used in such corpora as the Wall Street Journal and Brown corpora, which themselves are used to train conventional NLP techniques, so the ability to convert documents from other *noisy* forms of text provides an opportunity to apply existing techniques to text where they have not been applicable before.

Text normalization itself presents significant computational challenges, however. The variety among and within noisy forms, the cost of creating annotated corpora and resultant scarcity thereof, and the ambiguity when mapping some noisy corpora such as unedited text to an equivalent normal form make it a difficult task. These challenges may be why previously there has been no general approach to text normalization. In previous work, techniques for applying text normalization treated text normalization as a collection of different unrelated tasks, largely ignoring the overlap between them.

This dissertation identifies the commonalities between prior techniques in text normalization, and describes a technique that can treat what once were considered different tasks entirely as simply variants within a single unified task. The ability to apply a model to different sources of noise without needing to develop a new architecture or set of features allows us to save engineering time and effort. Our work focuses primarily on lexical normalization, or the normalization of words themselves, rather than their ordering, and is centered around the capability of the relatively new field of deep learning to achieve this level of generalization.

In initial analyses, we used a simple feed-forward neural architecture to normalize Twitter data without feature engineering, achieving a close third place in an international competition on lexical normalization and demonstrating the potential of deep learning for the task (Chapter 3). A subsequent larger scale analysis added a bidirectional gated recurrent neural network for inclusion of context as well as expanding the task to include three different sources of noisy data, adding optical character recognition post-processing (OCR) and automatic speech recognition post-processing (ASR). We found that our approach, DeepNorm, was competitive

with our competing models on all three corpora. On the OCR corpus and the Twitter corpus, it significantly outperformed the specialized approach when applied to the source of noise for which the approach was not specialized (Chapter 5). We have confirmed that DeepNorm can function effectively across multiple domains where the specialized techniques native to these domains cannot. This work advances the state of the art in lexical normalization by eliminating the need for manual feature engineering to optimize an architecture for the correction of different sources of noise.

REFERENCES

- At Aw, Min Zhang, Juan Xiao, and Jian Su. 2006. A Phrase-based Statistical Model for SMS Text Normalization. *Proceedings of the Association for Computational Linguistics*:33–40.
- Mayce Al Azawi, Adnan Ul Hazan, Marcus Liwicki, and Thomas M. Breuel. 2014. Character-Level Alignment Using WFST and LSTM for Post-processing in Multi-script Recognition Systems - A Comparative Study. In *International Conference on Image Analysis and Recognition*, volume 8814, pages 379–386.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*, pages 1–15.
- Timothy Baldwin, Marie Catherine De Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. 2015. Shared Tasks of the 2015 Workshop on Noisy User-generated Text : Twitter Lexical Normalization and Named Entity Recognition. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 126–135, Beijing, China.
- Youssef Bassil and Mohammad Alwani. 2012. Ocr Post-processing Error Correction Algorithm Using Google Online Spelling Suggestion. *arXiv preprint arXiv:1204.0191*, 3(1).
- Youssef Bassil and Paul Semaan. 2012. ASR Context-Sensitive Error Correction Based on Microsoft N-Gram Dataset. In *arXiv preprint arXiv:1203.5262*, volume 4, pages 34–42.
- Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. 2012. Theano: New Features and Speed Improvements. In *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*, pages 1–10. November.
- Yoshua Bengio. 2009. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127.
- Yoshua Bengio and Jean Sébastien Senécal. 2008. Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model. *IEEE Transactions on Neural Networks*, 19(4):713–722.
- James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-farley, and Yoshua Bengio. 2010. Theano: A CPU and GPU Math Compiler in Python. In *Proceedings of the 9th Python in Science Conference*, pages 3–10, Austin, Texas.
- George Box and George Tiao. 1992. *Bayesian Inference in Statistical Analysis*. Wiley, New York.
- Eric Brill and Robert C. Moore. 2000. An Improved Error Model for Noisy Channel Spelling Correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293.

- Peter F. Brown, Peter V. DeSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jennifer C. Lai. 1992. Class-based N-gram Models of Natural Language. *Computational Linguistics*, 18:467–479.
- E Byambakhishig, K Tanaka, R Aihara, T Nakashika, T Takiguchi, and Y Arika. 2014. Error Correction of Automatic Speech Recognition Based on Normalized Web Distance. In *Fifteenth Annual Conference of the International Speech Communication Association*, pages 2852–2856.
- Kyunghyun Cho, Bart Van Merriënboer, and Dzmitry Bahdanau. 2014a. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. *arXiv preprint arXiv:1409.1259*.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*.
- Monojit Choudhury, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar, and Anupam Basu. 2007. Investigation and Modeling of the Structure of Texting Language. *International Journal on Document Analysis and Recognition*, 10(3-4):157–174.
- Grzegorz Chrupala. 2014. Normalizing Tweets with Edit Scripts and Recurrent Neural Embeddings. *Association for Computational Linguistics*:680–686.
- Grzegorz Chrupala. 2013. Text Segmentation with Character-level Text Embeddings. In *The 30th International Conference on Machine Learning*.
- Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv*:1–9.
- Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. In *Journal of Machine Learning Research*, volume 12, pages 2493–2537.
- Paul Cook and Suzanne Stevenson. 2009. An Unsupervised Model for Text Message Normalization. *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*(June):71–78.
- Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. :1370–1380.
- Arthur Conan Doyle. 1892. *The Adventures of Sherlock Holmes*. George Newnes.
- Jacob Eisenstein. 2013. What to do About Bad Language on the Internet. In North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 359–369.
- Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-antoine Manzagol, and Pascal Vincent. 2010. Why Does Unsupervised Pre-training Help Deep Learning? *The Journal of Machine Learning Research*, 11:625–660.

- Julius Goth. 2012. Intrasentential Grammatical Correction with Weighted Finite State Transducers. Ph.D. thesis, North Carolina State University.
- Stephan Gouws, Dirk Hovy, Donald Metzler, and Marina Rey. 2011. Unsupervised Mining of Lexical Variants from Noisy Text. In *Conference on Empirical Methods in Natural Language Processing*, pages 82–90.
- M Gutmann and A Hyvärinen. 2010. Noise-contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models. In *International Conference on Artificial Intelligence and Statistics*, pages 1–8.
- Martin Hacker and Elmar Noth. 2014. A Phonetic Similarity Based Noisy Channel Approach to ASR Hypothesis Re-Ranking and Error Detection. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 2336–2340.
- Bo Han. 2014. Improving the Utility of Social Media with Natural Language Processing. Ph.D. thesis, University of Melbourne.
- Bo Han and Timothy Baldwin. 2011. Lexical Normalisation of Short Text Messages : Mkn Sens a # Twitter. In *Computational Linguistics*, pages 368–378.
- Bo Han, Paul Cook, and Timothy Baldwin. 2012. Automatically Constructing a Normalisation Dictionary for Microblogs. In *Empirical Methods for Natural Language Processing*, pages 421–432.
- Geoffrey Hinton. 2010. A Practical Guide to Training Restricted Boltzmann Machines A Practical Guide to Training Restricted Boltzmann Machines. In *Neural Networks: Tricks of the Trade*, volume 9, pages 1–20.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.
- Stéphane Huet, Guillaume Gravier, and Pascale Sébillot. 2010. Morpho-syntactic post-processing of N-best Lists for Improved French Automatic Speech Recognition. *Computer Speech and Language*, 24(4):663–684.
- Quoc V. Le Ilya Sutskever, Oriol Vinyals. 2014. Sequence to Sequence Learning with Neural Networks. In *Neural Information Processing Systems*, pages 1–9, Montreal, Canada.
- S. Impedovo, L. Ottaviano, and S. Occhinegro. 1991. Optical Character Recognition — A Survey. *International Journal of Pattern Recognition and Artificial Intelligence*, 05:1–24.
- Sebastien Jean, Roland Memisevic, Kyunghyun Cho, and Yoshua Bengio. 2015. On Using Very Large Target Vocabulary for Neural Machine Translation. *arXiv preprint arXiv:1412.2007*.
- Ning Jin. 2015. NCSU-SAS-NING: Candidate Generation and Feature Engineering for Supervised Lexical Normalization. In *Workshop for the Normalization of Noisy User Text*, pages 87–92, Beijing, China.
- Daniel Jurafsky and James H. Martin. 2008. *Speech and Language Processing (2nd Edition)*. Pearson Prentice Hall, New York, New York, USA.

- Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*:1700–1709.
- Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. *Association for Computational Linguistics*:655–665.
- Gabor Kata and Benoit Sagot. 2014. Automated Error Detection in Digitized Cultural Heritage Documents. In *EACL 2014 Workshop on Language Technology for Cultural Heritage*, pages 56–61.
- Max Kaufmann. 2010. Syntactic Normalization of Twitter Messages. In *International Conference on Natural Language Processing*, volume 2, pages 1–7, Kharagpur, India.
- Mark D. Kernighan, Kenneth W. Church, and William A. Gale. 1990. A Spelling Correction Program Based on a Noisy Channel Model. In *Proceedings of the Association for Computational Linguistics*, pages 205–210.
- Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-Aware Neural Language Models. In *arXiv preprint arXiv:1508.06615*.
- Catherine Kobus, François Yvon, and Géraldine Damnati. 2008. Normalizing SMS: Are Two Metaphors Better Than One? *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*(August):441–448.
- Okan Kolak and Philip Resnik. 2005. OCR Post-processing for Low Density Languages. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 867–874.
- Thomas K. Landauer and Peter W Foltz. 2012. An Introduction to Latent Semantic Analysis. *Discourse Processes*(April 2012):37–41.
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature*, 521(7553):436–444.
- Samuel Leeman-Munk, James Lester, and James Cox. 2015. NCSU_SAS_SAM: Deep Encoding and Reconstruction for Normalization of Noisy Text.
- Vladimir Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8):707–710.
- Fei Liu, Fuliang Weng, and Xiao Jiang. 2012. A Broad-Coverage Normalization System for Social Media Language. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics Volume 1 Long Papers*(July):1035–1044.
- Rafael Llobet, J. Ramon Navarro-Cerdan, Juan Carlos Perez-Cortes, and Joaquim Arlandis. 2010. OCR Post-processing Using Weighted Finite-state Transducers. In *Proceedings of the International Conference on Pattern Recognition*, pages 2021–2024.
- Liu Lon-Mu, Wei Sun Yair M. Babad, and Chan Ki-Kan. 1991. Adaptive Post-Processing of OCR Text Via Knowledge Acquisition. In *Proceedings of the 19th annual conference on Computer Science*, pages 558–569.

- Marco Lui and Timothy Baldwin. 2012. langid.py: An Off-the-shelf Language Identification Tool. *Proceedings of the ACL 2012 System Demonstrations*(July):25–30.
- Minh-thang Luong. 2015. Addressing the Rare Word Problem in Neural Machine Translation. In *Proceedings of the Association for Computational Linguistics*, pages 11–19, Beijing, China.
- Christopher D Manning and Hinrich Schütze. 1999. *Foundations of Natural Language Processing*.
- Andrei Mikheev. 2000. Document Centered Approach to Text Normalization. 23rd annual international ACM SIGIR conference on Research and development in information retrieval:136–143.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*:1–12.
- Tomas Mikolov, Ilya Sutskever, and Anoop Deoras. 2012. Subword Language Modeling with Neural Networks. *Preprint (Http://Www. Fit. ...)*:1–4.
- Wookhee Min and Bradford W Mott. 2014. NCSU_SAS_WOOKHEE : A Deep Contextual Long-Short Term Memory Model for Text Normalization. In *Proceedings of the Workshop for the Normalization of Noisy User Text*, pages 111–119.
- Andriy Mnih and Koray Kavukcuoglu. 2013. Learning Word Embeddings Efficiently with Noise-contrastive Estimation. *Neural Information Processing Systems*:1–9.
- Jordan Novet. 2015. Google Says its Speech Recognition Technology now has Only an 8% Word Error Rate.
- Olutobi Owoputi, Brendan O’Connor, Chris Dyer, Kevin Gimpel, and Nathan Schneider. 2012. Part-of-Speech Tagging for Twitter: Word Clusters and Other Advances. *Cmu-MI-12-107*.
- Deana Pennell and Yang Liu. 2011a. A Character-Level Machine Translation Approach for Normalization of SMS Abbreviations. *Proceedings of 5th International Joint Conference on Natural Language Processing*(2007):974–982.
- Deana Pennell and Yang Liu. 2011b. Toward Text Message Normalization: Modeling Abbreviation Generation. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*:5364–5367.
- Jeffrey Pennington and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods for Natural Language Processing*.
- Mohammad Pezeshki. 2015. Sequence Modeling using Gated Recurrent Neural Networks. *arXiv preprint arXiv:1501.00299*:1–7.
- Lev Ratinov and Joseph Turian. 2010. Word representations : A Simple and General Method for Semi-supervised Learning. In *Association for Computational Linguistics*, pages 384–394.

- Eric K. Ringger and James F. Allen. 1996. A Fertility Channel Model for Post-correction of Continuous Speech Recognition. *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*, 2.
- David Rumelhart, Geoffrey Hinton, and Ronald Williams. 1986. Learning Representations by Back-propagating Errors. *Nature*, 323(9):533–536.
- Arup Sarma and David D. Palmer. 2004. Context-based Speech Recognition Error Detection and Correction. *Proceedings of HLT-NAACL 2004*, 2004:85–88.
- Tim Schlippe, Chenfei Zhu, Jan Gebhardt, and Tanja Schultz. 2010. Text Normalization Based on Statistical Machine Translation and Internet User Support. *Interspeech*(September):1816–1819.
- Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.
- Claude E. Shannon. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423.
- Maria Shugrina. 2010. Formatting Time-Aligned ASR Transcripts for Readability. In *The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 198–206.
- Richard Sproat, Alan W. Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. 2001. Normalization of Non-standard Words. *Computer Speech & Language*, 15(3):287–333.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15:1929–1958.
- Kazem Taghva and Eric Stofsky. 2001. OCRSpell: An Interactive Spelling Correction System for OCR Errors in Text. *International Journal on Document Analysis and Recognition*, 3(3):125–137.
- Xiang Tong and David Evans. 1996. A Statistical Approach to Automatic OCR Error Correction in Context. In *Proceedings of the Fourth Workshop on Very Large Corpora*.
- Kristina Toutanova, Dan Klein, and Christopher D. Manning. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. (June):173–180.
- Kristina Toutanova and Robert C. Moore. 2002. Pronunciation Modeling for Improved Spelling Correction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, number July, pages 144–151.
- Johannes Twiefel, Timo Baumann, Stefan Heinrich, and Stefan Wermter. 2014. Improving Domain-independent Cloud-based Speech Recognition with Domain-dependent Phonetic Post-processing. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1–7, Quebec, Canada.
- Twitter. 2016. Twitter Usage Statistics - Internet Live Stats.

- Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-antoine Manzagol. 2008. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine learning - ICML '08*, pages 1096–1103.
- Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and Tell: A Neural Image Caption Generator. *arXiv preprint arXiv:1411.4555*.
- Robert A. Wagner and Michael J. Fischer. 1974. The String-to-String Correction Problem.
- Zhenzhen Xue, Dawei Yin, and Brian D. Davison. 2011. Normalizing Microtext. In *Analyzing Microtext*, pages 74–79.
- Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How Transferable are Features in Deep Neural Networks? In *Advances in Neural Information Processing Systems 27*, pages 1–9.
- Matthew D. Zeiler, Marc’Aurelio Ranzato, Rajat Monga, Mark Z. Mao, K. Yang, Quoc Viet Le, Patrick Nguyen, Andrew W. Senior, Vincent Vanhoucke, Jeffrey Dean, and Geoffrey E. Hinton. 2013. On Rectified Linear Units for Speech Processing. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*:3517–3521.
- Gang Zi and David Doermann. 2004. Document Image Ground Truth Generation from Electronic Text. *Proceedings of the 17th International Conference on Pattern Recognition*, 2:1–4.

APPENDICES

CHAPTER 8 APPENDICES

alpha	EXPN	abbreviation	<i>adv, N.Y, mph, gov't</i>
	LSEQ	letter sequence	<i>CIA, D.C, CDs</i>
	ASWD	read as word	<i>CAT, proper names</i>
	MSPL	misspelling	<i>geogaphy</i>
	NUM	number (cardinal)	<i>12, 45, 1/2, 0-6</i>
	NORD	number (ordinal)	<i>May 7, 3rd, Bill Gates III</i>
	NTEL	telephone (or part of)	<i>212 555-4523</i>
	NDIG	number as digits	<i>Room 101</i>
N	NIDE	identifier	<i>747, 386, 15, pc110, 3A</i>
U	NADDR	number as street address	<i>5000 Pennsylvania, 4523 Forbes</i>
M	NZIP	zip code or PO Box	<i>91020</i>
B	NTIME	a (compound) time	<i>3-20, 11:45</i>
E	NDATE	a (compound) date	<i>2/2/99, 14/03/87 (or US) 03/14/87</i>
R	NYER	year(s)	<i>1998, 80s, 1900s, 2003</i>
S	MONEY	money (US or other)	<i>\$3.45, HK\$300, Y20,000, \$200K</i>
	BMONEY	money tr/m/billions	<i>\$3.45 billion</i>
	PRCT	percentage	<i>75%, 3.4%</i>
	SPLT	mixed or "split"	<i>WS99, x220, 2-car</i> (see also SLNT and PUNC examples)
M	SLNT	not spoken, word boundary	<i>M.bath, KENT*RLTY, _really_</i>
	PUNC	not spoken, phrase boundary	non-standard punctuation: "****" in <i>\$99,9K***Whites, "..."</i> in <i>DECIDE... Year</i>
I			
S			
C	FNSP	funny spelling	<i>sllooooooww, sh*t</i>
	URL	url, pathname or email	<i>http://apj.co.uk, /usr/local, phj@tpt.com</i>
	NONE	should be ignored	ascii art, formatting junk

Appendix 1: An extensive taxonomy of non-standard words (Sproat et al., 2001)