Structured Abbreviation Expansion in Context

Kyle Gorman, Christo Kirov, Brian Roark, Richard Sproat
Google Inc.

Abstract

Ad hoc abbreviations are commonly found in informal communication channels that favor shorter messages. We consider the task of reversing these abbreviations in context to recover normalized, expanded versions of abbreviated messages. The problem is related to, but distinct from, spelling correction, in that ad hoc abbreviations are intentional and may involve substantial differences from the original words. Ad hoc abbreviations are productively generated on-the-fly, so they cannot be resolved solely by dictionary lookup. We generate a large, open-source data set of ad hoc abbreviations. This data is used to study abbreviation strategies and to develop two strong baselines for abbreviation expansion.

1 Introduction

Text normalization refers to transformations used to prepare text for downstream processing. Originally, this term was reserved for transformations mapping between "written" and "spoken" forms required by technologies like speech recognition and speech synthesis (Sproat et al., 2001), but it is now used for many other transformations, including normalizing informal text genres found on mobile messaging and social media platforms (e.g., Eisenstein, 2013; van der Goot, 2019).

Spans of text may require different kinds of normalization depending on their *semiotic class* (Taylor, 2009) and the requirements of the downstream application. For example, cardinal numbers such as *123* need to be normalized to a spoken form (*one hundred twenty three*) for speech processing, but this is not necessary for many text processing applications. The class of abbreviations has received particular attention. High-frequency, highly-conventionalized abbreviations, like those used for units of measure (e.g., *mL*, *lbs*) or geographic entities (e.g., *AK*, *NZ*) are often expanded using hand-written grammars (e.g., Ebden and Sproat, 2015), possibly augmented with machine learning systems for contextual disambiguation (e.g., Ng et al., 2017; Zhang et al., 2019).

In this study we are interested in a different subclass of abbreviations, those which are neither frequent nor conventionalized. We refer to these as *ad hoc abbreviations*. Such abbreviations are particularly common on those communication channels which demand or favor brevity, such as mobile messaging and social media platforms (Crystal, 2001, 2008; McCulloch, 2019). Unlike conventionalized abbreviations, ad hoc abbreviations are an open class, generated on-the-fly.

Unfortunately, there is little annotated data available to study ad hoc abbreviations as they occur in natural text. To remedy this, we provide a new open-source data set—derived from English Wikipedia—designed specifically to collect sentences with ad hoc abbreviations. We also provide two strong baseline abbreviation expansion systems, one finite-state, one neural, and find that abbreviations in context can be expanded with human-like accuracy. Both baselines use a noisy channel approach, which combines an abbreviation model, applied independently to each word in the sentence, and a language model enforcing fluency and local coherence in the expansion.

1.1 Contributions

The contributions of this study are three-fold. First, we describe a large data set for English abbreviation expansion made freely available to the research community. Secondly, we validate this data set using exploratory data analysis to identify common abbreviation strategies used in the training portion of the data set. Third, we describe and evaluate two strong baseline models—one using weighted finite-state transducers, the other using neural networks—and conduct ablation experiments and manual error analyses to study the relative contributions of our various design choices.

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 995–1005
November 7–11, 2021.

the	reason	i	went	to	the	store	was	to	buy	milk	and	bread	.
th	rsn	i	went	to	the	str	ws	to	buy	mlk	and	brd	.

Figure 1: Example of paired data for this task; above: expanded sequence; below: abbreviated sequence. Note that the data has been case-folded.

1.2 Related work

Text normalization was first studied for text-to-speech synthesis (TTS); Sproat et al. (2001) and van Esch and Sproat (2017) provide taxonomies of semiotic classes important for speech applications. Normalization for this application remains a topic of active research (e.g., Ebden and Sproat, 2015; Ritchie et al., 2019; Zhang et al., 2019). Roark and Sproat (2014) focus on abbreviation expansion and enforce a high-precision operating point, since, for TTS, incorrect expansions are judged more costly than leaving novel abbreviations unexpanded.

Abbreviation expansion has also been studied using data from SMS (Choudhury et al., 2007; Beaufort et al., 2010), chatrooms (Aw and Lee, 2012), and social media platforms such as Twitter (Chrupała, 2014; Baldwin et al., 2015). Most of the prior studies use small, manually curated databases in which "ground truth" labels were generated by asking annotators to expand the abbreviations using local context. Han and Baldwin (2011), Yang and Eisenstein (2013), and the organizers of the W-NUT 2015 shared task (Baldwin et al., 2015) all released data sets containing English-language Tweets annotated with expansions for abbreviations. Unfortunately, none of these data sets are presently available.[1] We are unaware of any large, publicly-available data set for abbreviation expansion, excluding a synthetic, automatically-generated data set for informal text normalization (Dekker and van der Goot, 2020).

A wide variety of machine learning techniques have been applied to abbreviation expansion, including hidden Markov models, various taggers and classifiers, generative and neural language models, and even machine translation systems. In this work we focus on supervised models, though unsupervised approaches have also been proposed (e.g., Cook and Stevenson, 2009; Liu et al., 2011; Yang and Eisenstein, 2013). The noisy channel paradigm we use to build baseline models here is inspired by earlier models for contextual spelling correction (e.g., Brill and Moore, 2000).

[1] This may reflect licensing issues inherent to Twitter data. This lead us to focus on sources with less restrictive licenses.

1.3 Task definition

We assume the following task definition. Let $\mathbf{A} = [a_0, a_1, \ldots, a_n]$ be a sequence of possibly-abbreviated words and let $\mathbf{E}$ be a sequence of expanded words $[e_0, e_1, \ldots, e_n]$, both of length n. If e_i is an element of $\mathbf{E}$, then the corresponding element of $\mathbf{A}$, a_i, must either be identical to e_i (in the case that it is not abbreviated), or a proper, non-null subsequence of e_i (in the case that it is an abbreviation of e_i). At inference time, the system is presented with an abbreviated $\mathbf{A}$ sequence of length n and is asked to propose a single hypothesis expansion of length n, denoted by $\hat{\mathbf{E}}$.

This formulation limits us to abbreviations that are derived via character deletion, and forbids pairs such as *because* → *cuz*, which can only be generated via insertion or substitution. In other words, this task corresponds to what Pennell and Liu (2010) call *deletion-based abbreviation*, arguably the most canonical form of abbreviation in English (e.g., Cannon, 1989). Furthermore, by asserting that the abbreviated sentence have the same number of words as the expanded sentence, we forbid mappings that involve multiple abbreviated or expanded tokens (e.g., *to be* → *2b*). These restrictions yield a highly-tractable task definition, but we anticipate that such restrictions can easily be relaxed in future work if desired.

2 Data

Our goal was to construct a data set consisting of English abbreviated/expanded sentence pairs as shown in Figure 1. In much of the previous work, these were created by finding text that contains likely abbreviations, and then asking annotators to disambiguate. However, Baldwin et al. (2015) reports that this disambiguation task results in poor inter-annotator agreement. Therefore, we instead choose to generate data using a task in which annotators *generate* abbreviated text rather than disambiguate it. Furthermore, since there are many ways to abbreviate any given sentence, one can easily collect multiple annotations per sentence.

2.1 Data set construction

We begin with sentences sampled from English-language Wikipedia pages. We then apply several filters to arrive at sentences in the collection that would fit the abbreviation generation elicitation approach. As detailed below, annotators are asked to delete at least a minimum number of characters from the sentence while preserving overall intelligibility. We select sentences of moderate length containing frequent words, avoiding proper names, technical jargon, and numerical expressions. Specific filters used to select sentences annotation are:

- Sentence length < 150 characters.
- Number of words in the sentence > 8.
- Average word length in the sentence ≥ 6.
- Sentence must match the regular expression `/^[A-Za-z',\-\ ]+\.$/`; i.e., the sentence must consist solely of Latin script letters, whitespace, and a few punctuation marks, including a sentence-final period.
- All non-initial words must be lowercase tokens (i.e., to avoid sampling proper names).

These filters produce a smaller corpus of roughly 4m sentences and 670k wordtypes. From this we construct a lexicon of roughly 100k words by retaining all those that occur at least 8 times, and remove any sentences that contain out-of-vocabulary tokens (OOVs). This set of preserved common words still contains many odd words and highly-specialized vocabulary. We therefore train a byte 5-gram language model from the data and use it to rank sentences by per-character entropy. We finally randomly sample 27k sentences with below-median per-character entropy for annotation, retaining another 2.7m sentences for language model training.

2.2 Human abbreviation generation

Ever since the earliest written texts, scribes have used ad hoc abbreviations to minimize space and time. Indeed, anyone who has studied ancient inscriptions is struck by the extremely high rate of abbreviation in such texts. For example, 11 of the 35 tokens in the dedicatory inscription at the base of Trajan's Column, completed in 113 CE, are—largely ad hoc—abbreviations (Figure 2).

With this in mind, we designed an annotation task in which a team of six in-house professional annotators, each working independently, were instructed to remove at least 20 characters from each sentence while maintaining overall intelligibility. A custom browser-based annotation interface is used to enforce the task limitations described in subsection 1.3, namely that abbreviations can only be produced by deletion and that no token can be totally deleted. It was expected that this annotation procedure would produce high rates of ad hoc abbreviation use—higher than is likely to occur naturally—and that subsequent expansion could be made less challenging by replacing a random fraction of the abbreviated tokens with their corresponding expansions, creating a corpus with a lower rate of abbreviation and reducing overall ambiguity. Annotators are provided no information about the intended use of this corpus.

SENATVS POPVLVSQVE ROMANVS
IMP CAESARI DIVI NERVAE F NERVAE
TRAIANO AVG GERM DACICO PONTIF
MAXIMO TRIB POT XVII IMP VI COS VI P P
AD DECLARANDVM QVANTAE ALTITVDINIS
MONS ET LOCVS TAN[tis oper]IBVS SIT EGESTVS

Figure 2: Latin dedicatory inscription at the base of Trajan's Column (*CIL* 6.960; Henzen et al., 1876), commemorating the Roman victory in the Dacian Wars, with abbreviations underlined.

	# sentences	# tokens
Training	21,318	332,829
Development	2,665	41,757
Testing	2,665	41,730
LM data	2,657,826	41,573,540

Table 1: Summary statistics for the data set.

The abbreviated/expanded sentence pairs are then randomly partitioned into training (80%), development (10%), and testing (10%) sets. Summary statistics for the data set are given in Table 1. Note that some sentences in the training set are deliberately abbreviated by multiple annotators; these are considered separate sentences for the purposes of this table.

2.3 Exploratory analysis

To validate this novel annotation process we conducted an exploratory analysis focusing on common abbreviation patterns used by the annotators. As shown in Table 2, over 45% of the training set tokens are abbreviated, and just over half of

deletions	count	%
0	182,552	54.8
1	78,872	23.7
2	42,976	12.9
3	17,105	5.1
≥ 4	11,324	3.4

Table 2: Histogram giving the number of deletions per token in the training set.

these have just one character deleted. This suggests that annotators frequently choose to make small changes to many words rather than making more aggressive abbreviations of a smaller number of words. Beyond single-character deletions, the most common strategies involve the deletion of a (string) suffix or (orthographic) vowels, as shown in Table 3. Simply eliding all orthographic vowels—and preserving all consonants—in a word is the single most common specific strategy. This strategy accounts for over a quarter of all training set abbreviations. Over 80% of the abbreviations found in the training set preserve all consonants. These results broadly accord with our intuitions about abbreviation formation in English.

2.4 Human abbreviation expansion

To further validate the annotation process and to establish a human topline, a separate team of three in-house annotators, each working independently, attempted to expand abbreviated sentences from the test set. As was the case for abbreviation generation, task restrictions were enforced using a custom browser-based annotation interface. These results are presented below in section 6, but anticipating the findings there, the second group of annotators were able to recover the original sentence with a high degree of accuracy.

2.5 Release

We release all annotated data under the Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA) License, the same license used by Wikipedia itself.[2] The release includes training, development, and testing data in the form of text-format Protocol Buffers messages,[3] as well as instructions for deserializing these messages.

[2] https://github.com/google-research-datasets/WikipediaAbbreviationData

[3] https://developers.google.com/protocol-buffers

3 Generative story

We approach the problem of abbreviation expansion as an instance of the noisy channel problem that has been applied to a wide range of problems including speech recognition (Jelinek, 1997; Mohri et al., 2002) and spelling correction (Brill and Moore, 2000). We first describe the generative process that produces the abbreviated sentence:

1. First, generate the expanded sentence $\mathbf{E} = [e_0, e_1, \ldots, e_n]$

2. Then, generate $\mathbf{A} = [a_0, a_1, \ldots, a_n]$ such that each element a_i is either

   (a) a non-null proper subsequence of e_i (i.e., a_i abbreviates e_i), or

   (b) equivalent to e_i (i.e., $a_i = e_i$).

Given $\mathbf{A}$, which we assume has passed through this noisy channel, our goal is to recover $\mathbf{E}$. We can naturally express this as a conditional model using Bayes' theorem.

$$\hat{\mathbf{E}} = \operatorname*{argmax}_{\mathbf{E}} P(\mathbf{E} \mid \mathbf{A}) \quad (1)$$

$$= \operatorname*{argmax}_{\mathbf{E}} P(\mathbf{E}) \cdot P(\mathbf{A} \mid \mathbf{E}) \quad (2)$$

$P(\mathbf{E})$, the probability of the expanded sequence, is naturally expressed by a language model over such sequences. For $P(\mathbf{A} \mid \mathbf{E})$, we make the simplifying assumption that the abbreviation of each token—or indeed, whether it is abbreviated at all—is independent of all other tokens. This allows us to approximate $P(\mathbf{E} \mid \mathbf{A})$ as the product

$$P(\mathbf{A} \mid \mathbf{E}) = \prod_{i=0}^{n} P(a_i \mid e_i). \quad (3)$$

Under these assumptions, a model for abbreviation expansion is parameterized by an expansion language model $P(\mathbf{E})$ and a per-token abbreviation generation model $P(a \mid e)$.

Below we describe methods for constructing language models and abbreviation models and show how these are used to infer the expanded sentence for a sentence containing abbreviations.

strategy	example			%
delete final *e*	native	→	nativ	12.0
delete other final letter	jamming	→	jammin	2.3
delete 2 final letters	however	→	howev	0.6
delete 3 final letters	volume	→	vol	1.2
delete 4 final letters	develop	→	dev	1.6
(total)				17.6
delete all vowels	government	→	gvrnmnt	26.2
delete all but word-initial	unheard	→	unhrd	10.9
delete all but first vowel	municipal	→	muncpl	9.3
delete all but final vowel	testing	→	tsting	3.8
delete other vowel subsets	reviewers	→	rviewrs	18.1
(total)				68.3
delete all vowels & other	background	→	bkgrnd	3.7
delete duplicated consonants	accessible	→	acesible	2.0
delete non-duplicated consonants	meetings	→	meetins	1.2
other	often	→	ofn	7.3
(total)				14.2

Table 3: Percentages of the 150k training set abbreviations following three major abbreviation strategies: suffix deletion, vowel deletion, and other strategies.

4 Models

We propose two baseline systems for noisy-channel decoding, one that relies on weighted finite-state transducers and one that uses a neural network language model.

4.1 Finite-state pipeline

The finite-state pipeline is defined by two weighted finite state automata. The first is a conventional n-gram language model defining a probability distribution over expansions

$$P(\mathbf{E}) = \prod_{i=0}^{n} P(e_i \mid h_i) \tag{4}$$

where h_i, the expansion history, is a finite suffix of $e_0, \ldots, e_{i-1}$. The second term is represented by a type of weighted finite-state transducer known variously as a *joint multigram model*, *pair n-gram model*, or *pair language model* (pair LM). Such models have been used for grapheme-to-phoneme conversion (Bisani and Ney, 2008; Novak et al., 2016), transliteration (Hellsten et al., 2017; Merhav and Ash, 2018), and abbreviation expansion (Roark and Sproat, 2014) among other tasks.

A pair LM α is a joint model over input/output strings $P(a_i, e_i)$ where a_i is an abbreviation and e_i an expansion. To train the pair LM, one first uses expectation maximization or related algorithms to align the characters of an abbreviation to its expansion. For example, for the pair *brd* → *bread*, the alignment might be $[\texttt{b:b}, \texttt{r:r}, \varepsilon\texttt{:e}, \varepsilon\texttt{:a}, \texttt{d:d}]$ where ε stands in for the empty string. Then, these alignments are used to construct a conventional n-gram language model representing the joint probability over input/output pairs (e.g., $\texttt{b} : \texttt{b}$).[4]

This model is applied to an abbreviated sentence as follows.[5] First, the abbreviated sentence $\mathbf{A}$ is encoded as an unweighted acceptor, composed with the closure of the pair LM α, and the result is output-projected (here indicated by π_o).

$$\eta = \pi_0[\mathbf{A} \circ \alpha^*]. \tag{5}$$

An example of the resulting lattice is shown in Figure 3. λ is an unweighted transducer in which each path maps an in-vocabulary word, encoded as a

[4] Computing the conditional probability $P(a_i \mid e_i)$ in eq. 3 from the joint probability $P(a_i, e_i)$ requires a computationally expensive summation over all possible alignments. However we find it can be effectively approximated using the most probable alignment according to the joint probability model.

[5] We assume the reader is familiar with finite-state automata and algorithms such as composition, concatenation, projection, and shortest path. See Mohri 2009 for a review of finite-state automata and these algorithms.

character sequence, to that same word encoded as a single symbol, as is done in the expansion LM. To construct the final lattice, η is composed with the closure of λ, output-projected, and composed with the expansion language model μ. The best expansion is given by

$$\hat{\mathbf{E}} = \text{ShortestPath}[\pi_o[\eta \circ \lambda^*] \circ \mu], \qquad (6)$$

the shortest path through a weighted lattice of candidate expansions.

4.2 Neural pipeline

The neural pipeline replaces the n-gram language model with a recurrent neural network language model. Unlike conventional n-gram models, recurrent neural models do not in principle impose an upper bound on the amount of context used to condition their predictions. While the pair LM can be fused with a neural LM, an alternative abbreviation model was also considered. As before, we wish to generate a set of expansion candidates and allow the language model to select the best candidate in context. Any in-vocabulary word is considered to be a candidate for e_i so long as it is a supersequence of a_i. A list of such candidates can be generated by constructing a transducer ν that allows for identity or insertion relations. Then

$$\pi_o[a_i \circ \nu \circ \lambda] \qquad (7)$$

contains all possible expansions of a_i.

We assign weights to the operations of ν. Identity mappings are given zero cost, whereas insertion costs are given by the negative log probability of that character's insertion, estimated using maximum likelihood estimation on the training set. Probabilities for initial and final insertions are computed separately from word-internal insertions. These weights allow the system to rank candidate expansions at each position. Only the 8 best candidates are considered at each position, and candidates whose path weights are more than twice the cost of the best candidate path are pruned. A few additional heuristics are used to represent the tension between brevity and fidelity.

- **LexBlock:** If a_i is in-vocabulary, set the probability of all other output candidates to zero.
- **Memory:** Do not prune an expansion candidate e_i if it is occurs as an expansion of a_i in the training set.
- **SubBlock:** If one candidate is a contiguous substring of another, set the probability of the superstring candidate to zero. For example, for the abbreviation *ct*, this heuristic will discard the candidate *cats* in favor of *cat*.

We refer to this as the *subsequence model* to contrast it with the pair LM proposed earlier.

Decoding of the neural pipeline is similar to that of the finite-state pipeline with the addition of pruning and the optional application of the above heuristics. However, finding the highest-probability path according to the neural language model is somewhat more challenging than is the case for the finite-state model. Because there is no upper bound on the amount of context used by the neural language model, the score for each node of the lattice depends on the full path taken to reach that node, and the decoding graph is a prefix tree of all paths through the lattice. As the number of such paths grows exponentially as a function of sentence length, left-to-right approximate beam search (Graves, 2012) with a beam of size 20 is used as an alternative to exhaustive search.

5 Experiments

We perform experiments with both pipelines described above, the finite state and neural pipelines described in section 4, and compare their performance with human participants attempting to expand the same abbreviated text. The training set described in section 2 is used to train the abbreviation models. Language models are trained using the concatenation of the training set with 2.7m additional sentences from Wikipedia as described in subsection 2.1. The development set was used to tune the Markov order of the finite-state components, and to ablate the subsequence model heuristics. Final evaluations are conducted on the test set. The full vocabulary consists of all 75k word-types appearing in the language model training set, simulating a general-domain normalization task.

5.1 Finite-state implementation

Expansion model The expansion model is a conventional language model over expansion tokens. The OpenGrm-NGram toolkit (Roark et al., 2012) is used to build a trigram model with Kneser-Ney smoothing (Ney et al., 1994) and ε-arcs used to approximate back-offs. It is then shrunk using relative entropy pruning (Stolcke, 1998).

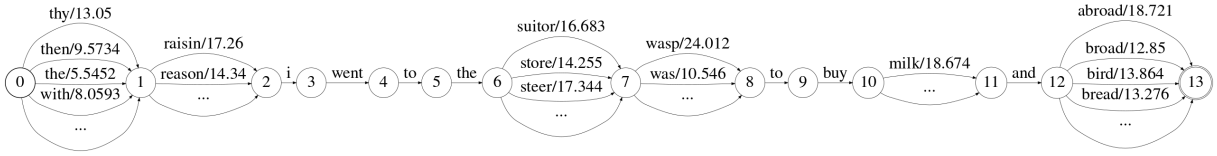

Figure 3: An example η lattice corresponding to the example sentence in Figure 1. Ellipses indicate that arcs have been pruned for reasons of space. Non-zero costs are indicated by negative log probability arc weights.

Abbreviation model The abbreviation model is a pair n-gram language model over input/output character pairs, encoded as a weighted transducer. The OpenGrm-BaumWelch toolkit and a stepwise interpolated variant of expectation maximization (Liang and Klein, 2009) are used to compute alignments between abbreviations and expansions. OpenGrm-NGram is then used to train a 4-gram pair LM. As with the expansion model, Kneser-Ney smoothing with ε-arc back-offs are used, but no shrinking is performed.[6]

5.2 Neural implementation

Expansion model The expansion model consists of an embedding layer of dimensionality 512 and two LSTM (Hochreiter and Schmidhuber, 1997) layers, each with 512 hidden units. Each sentence is padded with reserved start and end symbols. The model, implemented in TensorFlow (Abadi et al., 2016), is trained in batches of 256 until convergence using the Adam optimizer (Kingma and Ba, 2015) with $\alpha = .001$.

Abbreviation model The subsequence model parameters are computed over the training set via maximum likelihood estimation.

5.3 Complexity

The complexity of both pipelines is dominated by their decoding step. The finite-state pipeline's shortest-path computation has complexity of $O(n \log n)$, where n is the length of the sequence to be decoded. Beam search for the neural network pipeline has complexity of $O(n)$.

5.4 Evaluation

The primary metric used for system comparison is word error rate (WER), the percentage of incorrect words in the expansion. We also compute more specific statistics: overexpansion rate (OER), the percentage of words in the hypothesis expansion which were expanded but did not require expansion, underexpansion rate (UER), the percentage of words which required expansion but were not expanded, and incorrect expansion rate (IER), the percentage of words which both required and received expansion but which were were expanded incorrectly.[7] As Roark and Sproat (2014) argue, an ideal abbreviation expansion system should be "Hippocratic" in the sense that it does no harm to human interpretability, so it is particularly important to minimize OER and IER errors. Other metrics such as character-level edit distance and sentence error rate are also computed. However, they are closely correlated with WER and are therefore omitted below.

6 Results

6.1 Test results

Table 4 gives an overview of results across the different experimental conditions as well as the human topline results. The best overall performance is achieved by the neural pipeline combined with the subsequence abbreviation model. Presumably the neural pipeline benefits from the more expressive model of local context, and the subsequence model outperforms the pair LM.

6.2 Ablation results

To measure the importance of the three heuristics used in the subsequence model, a series of ablation experiments are performed on the development set; results are given in Table 5. These show that the best performance was achieved after all heuristics discussed in subsection 4.2 are applied to the subsequence model. The ablation experiments are repeated with a smaller "task vocabulary" containing the 15k wordtypes occuring in the abbreviation

[6] We conducted several other experiments that had negative or negligible results, including the use of other smoothing techniques, using ϕ-arcs to exactly encode language model back-offs, and a finite-state implementation of the subsequence model's LexBlock heuristic.

[7] Note that UER and IER are calculated using the total words that *should* be expanded as a denominator, whereas OER is calculated using the total number of words that should *not* be expanded as a denominator. As a result, WER is not merely the sum of OER, IER, and UER.

	WER	OER	UER	IER
n-gram LM, pair LM	2.90	0.00	2.13	4.08
LSTM LM, pair LM	1.41	0.39	0.19	2.35
LSTM LM, subseq.	**1.12**	0.40	0.20	1.74
Human topline	3.51	2.23	0.30	4.88

Table 4: Baseline results, with a human topline for comparison. WER: word error rate; OER: overexpansion rate; UER: underexpansion rate; IER: incorrect expansion rate.

	full vocab.	task vocab.
Subsequence	7.92	9.56
…+LexBlock	9.16	8.47
…+Memory	1.18	3.94
…+SubBlock	1.07	3.85

Table 5: Development set WER results for the ablation experiments for the subsequence model heuristics.

training corpus. One interesting phenomenon is the apparent rise in error rate when the LexBlock heuristic is used with a full vocabulary. This is primarily due to the fact that the full vocabulary already includes abbreviated function words. Without also applying the Memory heuristic, LexBlock requires these common abbreviations to remain unexpanded. However, this is no longer an issue when the task vocabulary is used in place of the full vocabulary.

6.3 Error analysis

Using the development set, we perform a qualitative analysis focusing on overexpansion and incorrect expansion errors made by the best-performing system, the neural LM with a subsequence abbreviation model. The examples below give the **source abbreviation**, the ✓ **target expansion**, and the ✗ **predicted expansion**. They also give the reader an idea of the difficulty of the abbreviation expansion task in the presence of a high rate of abbreviation. A manual inspection of the 400 errors produced suggests that roughly 40% could be classified as harmful in the sense that they substantially modify the meaning of the underlying sentence.

(1) the **{clases, ✓ classes, ✗ clashes}** cntinud nd th band strugld fr time to rite tgthr .

(2) anothr criticism is abt th absenc o a stndrd **{auditin, ✓ auditing, ✗ audition}** procedr .

Sometimes these errors are difficult to avoid given a highly ambiguous context for a short abbreviation, with multiple plausible expansions. A broader, multi-sentence context might help further disambiguate these cases but would naturally require a more complex language model. Furthermore, 39.8% of *all* errors are unavoidable due the aggressive candidate pruning in the best performing conditions. The expected candidate is not an option for the model in these cases, though the ablation results suggest this is a sensible trade-off to make. The remaining errors are largely benign and showed several re-occurring patterns. One common problem is the model incorrectly choosing an unexpected American and or British spelling variant—both are present on Wikipedia—an easily-fixed inconsistency in the data.

(3) consequently th village hs develpd a mor suburbn role than som o its **{neighbrs, ✓ neighbours, ✗ neighbors}** .

It is also common for short abbreviations to be incorrectly expanded to a morphologically-related variant of expected expansion,[8] or to a function word with comparable syntactic effect.

(4) they **{recog, ✓ recognized, ✗ recognize}** accomps by musicians frm th prev yr .

(5) **{th, ✓ the, ✗ this}** behavr s strengthnd by an automatc reinfrcng consequenc .

7 Ethical concerns

The proposed technology is intended as a component of other speech and language processing systems. We note that abbreviation expansion systems have some small potential for abuse beyond those of the larger systems they might be integrated into. For instance, this technology could be used to

[8] We note that Żelasko (2018) considers the problem of disambiguating abbreviations in Polish, a language with far richer inflectional morphology.

defeat abbreviation as a strategy for circumventing algorithmic state censorship.

The data is drawn from English Wikipedia text and was produced by a team of professional annotators based in the United States; its use to disambiguate abbreviations generated by other English-speaking communities would likely introduce bias.

8 Conclusions

We introduce a large, freely-available data set for ad hoc abbreviation expansion, describing the validating the annotation paradigm used to develop it. Using this data set, we find that ad hoc abbreviation expansion can be performed at human levels of accuracy using noisy channel models. The finite-state pipeline described above has been integrated as an optional module for Google text-to-speech synthesis engines.

In future work we will survey abbreviation and abbreviation expansion beyond English. It is expected that abbreviation strategies may differ substantially across languages and scripts. Indeed, while they are integral features of some languages, particularly in informal genres, others appear to use few if any abbreviations at all.

Acknowledgments

The authors thank Olivia Redfield for assistance with data collection, and Caterina Golner and Katherine Wang for their help with data cleaning and pilot experiments.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, …, and Xiaoqiang Zheng. 2016. TensorFlow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283.

Ai Ti Aw and Lian Hau Lee. 2012. Personalized normalization for a multilingual chat system. In *Proceedings of the ACL 2012 System Demonstrations*, pages 31–36.

Timothy Baldwin, Young-Bum Kim, Marie Catherine de Marneffe, Alan Ritter, Bo Han, 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 126–136.

Richard Beaufort, Sophie Roekhaut, Louise-Amélie Cougnon, and Cédrick Fairon. 2010. A hybrid rule/model-based finite-state framework for normalizing SMS messages. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 770–779.

Maximilian Bisani and Hermann Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451.

Eric Brill and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 286–293.

Garland Cannon. 1989. Abbreviations and acronyms in English word-formation. *American Speech*, 64(2):99–127.

Monojit Choudhury, Rahul Saraf, Vijit Jain, Sudesha Sarkar, and Anupam Basu. 2007. Investigation and modeling of the structure of texting language. *International Journal of Document Analysis and Recognition*, 10:157–174.

Grzegorz Chrupała. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 680–686.

Paul Cook and Suzanne Stevenson. 2009. An unsupervised model for text message normalization. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 71–78.

David Crystal. 2001. *Language and the Internet*. Cambridge University Press.

David Crystal. 2008. *Txtng: The Gr8 Db8*. Oxford University Press.

Kelly Dekker and Rob van der Goot. 2020. Synthetic data for English lexical normalization: how close can we get to manually annotated data? In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 6300–6309.

Peter Ebden and Richard Sproat. 2015. The Kestrel TTS text normalization system. *Natural Language Engineering*, 21(3):333–353.

Jacob Eisenstein. 2013. What to do about bad language on the internet. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 359–369.

Daan van Esch and Richard Sproat. 2017. An expanded taxonomy of semiotic classes for text normalization. In *Proceedings of INTERSPEECH*, pages 4016–4020.

Rob van der Goot. 2019. MoNoise: a multi-lingual and easy-to-use lexical normalization tool. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 201–206.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. Paper presented at the Representation Learning Workshop, ICML 2012.

Bo Han and Timothy Baldwin. 2011. Lexical normalisation of short text messages: makn sense a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 368–378.

Lars Hellsten, Brian Roark, Prasoon Goyal, Cyril Allauzen, Françoise Beaufays, Tom Ouyang, …, and David Rybach. 2017. Transliterated mobile keyboard input via weighted finite-state transducers. In *Proceedings of the 13th International Conference on Finite State Methods and Natural Language Processing*, pages 10–19.

Wilhelm Henzen, Eugen Bormann, and Giovanni Rossi Battista, editors. 1876. *Corpus Inscriptorum Latinarum: Inscriptiones Urbis Romae Latinae*, volume 6. Berolini.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Frederick Jelinek. 1997. *Statistical Methods for Speech Recognition*. MIT Press.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: a method for stochastic optimization. In *3rd International Conference on Learning Representations: Conference Track Proceedings*.

Percy Liang and Dan Klein. 2009. Online EM for unsupervised models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 611–619.

Fei Liu, Fuliang Weng, Bingqing Wang, and Yang Liu. 2011. Insertion, deletion, or substitution? Normalizing text messages without pre-categorization nor supervision. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 71–76.

Gretchen McCulloch. 2019. *Because Internet: Understanding the New Rules of Language*. Riverhead Books.

Yuval Merhav and Stephen Ash. 2018. Design challenges in named entity transliteration. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 630–640.

Mehryar Mohri. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 213–254. Springer.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38.

Axel H. Ng, Kyle Gorman, and Richard Sproat. 2017. Minimally supervised written-to-spoken text normalization. In *IEEE Automatic Speech Recognition and Understanding Workshop*, pages 665–670.

Josef Robert Novak, Nobuaki Minematsu, and Keikichi Hirose. 2016. Phonetisaurus: exploring grapheme-to-phoneme conversion with joint n-grams models in the WFST framework. *Natural Language Engineering*, 22(6):907–938.

Deana Pennell and Yang Liu. 2010. Normalization of text messages for text-to-speech. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4842–4845.

Sandy Ritchie, Richard Sproat, Kyle Gorman, Daan van Esch, Christian Schallhart, Bampounis Nikos, …, and Eoin Mahon. 2019. Unified verbalization for speech recognition & synthesis across languages. In *Proceedings of INTERSPEECH*, pages 3530–3534.

Brian Roark and Richard Sproat. 2014. Hippocratic abbreviation expansion. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369.

Brian Roark, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. 2012. The OpenGrm open-source finite-state grammar software libraries. In *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66.

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.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proceedings of the DARPA Broadcast News and Understanding Workshop*, pages 270–274.

Paul Taylor. 2009. *Text-to-Speech Synthesis*. Cambridge University Press.

Yi Yang and Jacob Eisenstein. 2013. A log-linear model for unsupervised text normalization. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 61–72.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337.

Piotr Żelasko. 2018. Expanding abbreviations in a strongly-inflected language: are morphosyntactic

tags sufficient? In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, pages 1880–1884.