INCLUDING PUNCTUATION, CAPITALIZATION AND NUMERAL IN AUTOMATIC SPEECH RECOGNITION

Xudong Liang (Brandon)

UNI: xl2891

ABSTRACT

In traditional Automatic Speech Recognition (ASR) systems, the generated transcript output is typically unsegmented text that includes only the plain word tokens and excludes other sentence components, such as punctuations, capitalized phrases and numerals, due to text normalization in preprocessing. Although such transcripts containing word tokens alone can provide sufficient translation of the speech audio, the other sentence components are also crucial to understanding the message. In this work, I propose modification in the lexicon, acoustic model, and language model components to a typical ASR architecture to include punctuation, capitalization and numeral in the predicted transcript. Specifically, I propose two distinct ways to include punctuation and compare both results to the Kaldi Ted-Lium model. Despite the expected higher Word Error Rate (WER) by the enhanced ASR models, the gap is not far off compared to the Ted-Lium model.

Keywords: Automatic Speech Recognition, Punctuation, Capitalization, Numeral

1. INTRODUCTION

In traditional Automatic Speech Recognition (ASR) systems, the generated transcript output typically only includes the plain word tokens, but not other sentence components, such as punctuations, capitalized phrases and numerals. These transcripts are also called unsegmented text [1]. This is normally due to text normalization in preprocessing. In this work, I focus on an end-to-end approach to restore these lost sentence components in the ASR transcript: punctuation, capitalization and numeral.

Punctuation refers to the marks used to separate sentences and clauses, such as comma, period and semi-colon; some punctuations not only provide intra-sentence separations, but also imply the underlying emotions, such as exclamation mark and question mark. **Capitalization** (or **capitalized phrase**) refers to the word phrases whose first letters are capitalized; this is always the case for the first word in a sentence and often the case for special nouns, such as "Michael Jackson" or "White House". As we can see, capitalization in ASR is thus instrumental in identifying named entities from a speech to its transcript, since "the white house" and "the White House" can convey drastically different meanings. **Numeral** refers to converting the spelled-out numerical words or phrases to the numerical quantities or discrete values they represent in numbers, such as year and phone number (e.g.: "eighteen twelve" to "1812"). A written numeral in a transcript can lead readers directly to the quantity or value of its corresponding entity or event, as opposed to spelling out the numbers in word, which often requires readers to translate to the numbers internally first.

Traditional ASR	what happened in eighteen twelve to the white house
Desired ASR	What happened in 1812 to the White House?

Table 1. A sample ASR transcript with plain word tokens only compared to its desired/actual transcript with the other sentence components.

Although a transcript containing word tokens alone can provide sufficient information of the speech audio, it would often obstruct readers from understanding the ASR output semantically and cause difficulties for downstream Natural Language Processing (NLP) models [2]. That being said, it is definitely better to include the three aforementioned sentence components, since they are also crucial to understanding the intrinsic relationships in the transcript (according to their definitions). Table 1 shows the difference between a traditional ASR transcript with only plain word tokens and a desired ASR transcript that includes the other sentence components. As we can see, including the punctuations, capitalized phrases and numerals in the desired format provides a more natural way of reading and understanding the speech transcript. With this motivation, this project focuses on including punctuation, capitalization and numeral in ASR transcript.

For consistency, in the remainder of the paper, I refer to the conventional ASR system that only generates plain word tokens as "Traditional ASR" and the updated ASR system that includes punctuation, capitalization and numeral as "Enhanced ASR". I also use ARPAbet-style phonetic transcription to spell out pronunciations in the paper.

Special thanks to Professor Beigi.

2. RELATED WORKS

In [3], the author uses the phonetic characteristics of punctuation marks, and properly processed lexical information for automatic generation of punctuation in speech transcripts. Despite implementation and application difference, this work inspires me to identify lexicon adjustment as the foundation to my approach in order to incorporate phonetic features to the acoustic model and updated N-Grams to the language model.

In [4], the authors use Bidirectional Long Short-Term Memory (Bi-LSTM) and Convolutional Neural Network (CNN) for punctuation prediction, where CNN yields a better precision and Bi-LSTM yields a better recall. Similarly, in [5], the authors also use LSTM for punctuation prediction and the authors in [2] use transformer and chunk merging (decoding in parallel) for capitalization and punctuation recovering. In [6], the authors use maximum entropy models and word part-of-speech for recovering punctuations. In [1], the authors use a Bidirectional Recurrent Neural Network (Bi-RNN) with attention mechanisms for punctuation restoration in unsegmented text. In [7], the authors use attention and encoder-decoder-based segmentation for punctuation insertion. Similarly, in [8], the authors treat the punctuation prediction as monolingual translation problem (source language being unpunctuated sentences and target language being punctuated) using segmentation. These works inspire me to incorporate Neural Network models to the acoustic modeling and language modeling components of the "Enhanced ASR" architecture for not only punctuation prediction but also capitalization and numeral, in addition to statistical Gaussian-Mixture Hidden Markov Model (GMM-HMM) for acoustic modeling and N-Gram for language modeling.

Regarding neural language model, in [9], the author discovers that using Transformer as language model in ASR actually yields less optimal results (reflected by WER) than the conventional RNN language model for Ted-Lium [10] in Table 1. The author reasons that it is most likely due to the large parameter count of Transformer (e.g.: XLNet) compared to the relatively small dataset. Therefore, considering the size of my dataset (mentioned in Section 3), I decide to use RNN language model instead of Transformer-based language model for the "Enhanced ASR" model.

In [11], the authors propose a sequence-to-sequence speech embedding framework that extends from word embedding. This speech embedding framework is able to learn fixed-length vector representations of audio segments. In [12] and [13], the authors use Conditional Random Field (CRF) [14] for punctuation prediction. Due to time constraint, I decide not to explore or incorporate these ideas.

It is worth mentioning that most of the related literature focuses on punctuation recovering, rather than the other two sentence components, which is another motivation for this project to achieve an end-to-end model for all three sentence components instead of just punctuation.

In addition, I mimic Kaldi's Ted-Lium recipe [15, 10] for the implementation of my workflow.

3. DATA

For this project, I use two datasets, of which the details are described below.

- **Ted-Lium**[1] [10]: This dataset contains 2,351 audio talks of TED speeches in NIST sphere format (SPH), of which the total duration is 452 hours. The transcripts of all talks are aligned automatically. It also provides the dictionary with pronunciations. The sampling rate is 16kHz and precision is 16 bit. Note that the transcript for this dataset does not include punctuation, capitalization or numeral.

- **National Speech Corpus (NSC)**[2] [16]: (Also known as **Singapore National Speech Corpus** dataset) This is a corpus of aligned spoken utterances in English, consisting of 826,469 short utterances (as opposed to long TED talks in Ted-Lium) with about 1,000 hours in total, from 1,031 local English speakers. Most importantly, this is the only dataset that provides punctuation, capitalization and numeral in the transcript.

Since **Ted-Lium** does not provide punctuation, capitalization or numeral, I use it to pre-train a "Traditional ASR" model (from Kaldi's Ted-Lium recipe [15, 10]), which is then fine-tuned on the **NSC** dataset for the "Enhanced ASR" model. The approach and training workflow are explained in detail in Section 4 below.

4. APPROACH & ARCHITECTURE

Since the pre-trained Ted-Lium model [10] (a "Traditional ASR") follows a pre-defined recipe [15], I skip its workflow in this section. Instead, in this section I focus on the workflow for the "Enhanced ASR".

Figure 1 illustrates the architecture of a "Traditional ASR" system. As we can see, it is broken down into several components with different dependencies. The modification for the "Enhanced ASR" takes places in the following four bolded regions in Figure 1: Lexicon, Acoustic Model, Language Model and Transcript (Data). For Acoustic Model and Language Model implementation, I mainly follow the workflow of Kaldi Ted-Lium recipe [15, 10], as explained in Section 4.2 and 4.3 below.

[1] https://www.openslr.org/51/

[2] https://www.imda.gov.sg/programme-listing/digital-services-lab/national-speech-corpus; I only use PART2 of this dataset since it is the only portion that provides punctuations, capitalizations and numerals in the transcript.

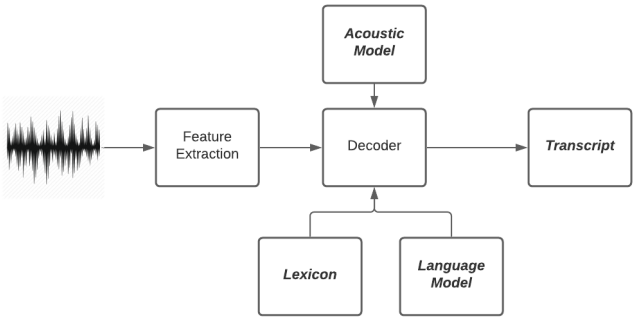

Fig. 1. The architecture of a traditional ASR system, where most modifications of this project take place in **Lexicon**, **Acoustic Model**, **Language Model** and **Transcript**.

4.1. Lexicon (text-to-phone mapping)

The most important component in ASR workflow is to provide an accurate lexicon dictionary for the model. A lexicon is essentially a dictionary mapping of each word to its ARPAbet-transcribed pronunciation in the entire vocabulary (e.g.: "seat": "S IH T"). Therefore, how to accurately and exhaustively include and distinguish pronunciations for punctuation, capitalization and numeral in the lexicon is the key to the success of the "Enhanced ASR" model.

I use the lexicon from Kaldi's Ted-Lium recipe [15, 10], which is all lower-cased and without punctuations and numerals, as the default lexicon dictionary, and apply the following additions to it.

1. **Capitalization**: This is perhaps the easiest addition since capitalization does not change the pronunciation of the original word. For each word token from the default lexicon, I duplicate its entry along with its pronunciation and capitalize the first letter (if the word starts with a letter). For example, for the word token "hello", I also include a token "Hello" with the same ARPAbet pronunciation. Thus, the capitalized words are counted as separate word tokens from their lower-cased normalized versions, but with the same pronunciations. This roughly doubles the size of the lexicon. Note that in this project, I discard the cases where the entire word token is capitalized (e.g.: "APEC"), because it is highly unlikely that NSC transcript includes such cases.

2. **Punctuation**: I consider two versions of including punctuations in the lexicon:

   (a) **Version 1 - Attach punctuations to the preceding tokens**: In common English text, a punctuation is normally attached to the end of its preceding word token (e.g.: comma, period, exclamation mark, etc.) with no space in between. This means, when considering punctuations in the updated lexicon, each word token (including the capitalized ones from above) has a possibility, no matter how unlikely, to attach any punctuation to its end. I append the punctuations to the lexicon by manually creating N (N being the number of unique punctuations) copies of all existing vocabulary (along with their pronunciation mapping) and attaching each of all possible punctuations in the end of each word token's copy. All these duplicates have the same pronunciations as the original token. For example, I duplicate the pronunciation mapping of "Hello" to "Hello,", "Hello!", etc. (for all possible punctuations). Using this approach, we are essentially implying to the acoustic model and language model of ASR system that with the same pronunciation, there are N possible options to consider for the same word (i.e. beginning or middle of the sentence, end of the sentence, before pause, etc.). The limitation of this setup is that each word token gets duplicated N times regardless, despite the extremely low likelihood of having punctuations attached for the majority of the vocabulary. This roughly increases the lexicon size by a factor of N times ($N = 20$), which drastically increases the vocabulary dimension of the "Enhanced ASR".

   (b) **Version 2 - Treat punctuations as standalone tokens**: The second version treats punctuations as standalone word tokens with silence as their pronunciations. This is a simple addition to the updated lexicon by appending each unique punctuation as a word token with a silence phone to the end of the updated lexicon, which has a negligible increase to the lexicon size. I use NLTK's WordPunctTokenizer[3] to tokenize the original transcript in NSC dataset so that each punctuation is also treated as a standalone token in language modeling. A potential drawback or suspicion is that whether the "Enhanced ASR" can accurately distinguish true silence and punctuation tokens when encountering the silence phones. Another limitation is that some punctuations may have non-silence pronunciations (a period or a dot can be pronounced as "point" or "dot"), but these are very rare edge cases.

Since these two versions of lexical punctuations are mutually exclusive, I eventually create two lexicons to compensate for each version. Note that the additions of capitalization and numeral lexicon mapping are exactly the same in both lexicon versions, and the only difference is how punctuation lexicons are treated. It would certainly be interesting to compare the ASR model re-

[3] https://www.nltk.org/api/nltk.tokenize.html

sults of these two lexicon versions. An early guess is that Version 2 would outperform Version 1 due to a significantly smaller lexicon size and the fact that standalone punctuations would provide less confusion in language modeling as opposed to Version 1.

3. **Numeral**: Unlike capitalization and punctuation above, which require minimal effort in pronunciation construction, numeral requires large amount of pronunciation construction as a new set of lexicon mapping. I then numeral lexicon pronunciations into the following two categories.

   (a) **Normal Pronunciation using Dynamic Programming**: This type of pronunciation refers to how we normally pronounce numbers. For example, "11" is expressed as "eleven" ("IH L EH V AH N"), "25" is "twenty five" ("T W EH N T IY F AY V"), "34" is "thirty four" ("TH ER D IY F OW R"), "73" is "seventy three" ("S EH V AH N T IY TH R IY) and "115" is "a/one hundred and fifteen" ("AH/W AH N HH AH N D R AH D AH N D F IH F T IY N"). While these pronunciations are intuitive in our mind, it is not plain simple in transcription. However, there exist certain patterns to follow. I first use CMU Pronouncing Dictionary [4] to generate the ARPAbet-transcribed pronunciations of the basic number units (e.g.: numbers below 20, 30, 40, the word "'a", 'hundred", "thousand", "million", "and", etc.). Then, I use these basic units to logically piece together the normal pronunciations for each number below 10,000. For numbers below 100, I combine the pronunciations of their decimal number units (20s, 30s, etc.) and those of their last-digit number. For example, the number 23 has the pronunciation of 20 followed by that of 3. Then for numbers below 1,000, I get the pronunciation of the hundred-digit number and that of the remaining two-digit number, and then insert the pronunciations of the word "hundred" and "and" in between. For example, the number 423 is expressed as "4 hundred and 23". Similar rules extend beyond 1,000 and it follows a dynamic programming approach to treat existing numeral pronunciations as lookup table. Table 2 shows some more examples of normal pronunciations for numerals. Note that for numbers that are larger than 99 and that start with the digit 1, their pronunciations can start with both "a" and "one" (e.g.: "one hundred" and "a hundred).

   (b) **Spelled-out (Naive) Pronunciation**: This subset of pronunciation treats each digit in a number as an independent pronunciation unit and therefore transcribes any given number digit by digit. For example, it transcribes "11" as "one one" ("W AH N W AH N"), while the normal pronunciation should be "eleven" ("IH L EH V AH N"). Despite the naive nature of this pronunciation style, it is still possible for people to pronounce numbers this way, such as spelling out phone numbers or a specific ID. So it is important to also include this type of number pronunciations in our numeral lexicon. To achieve this, I use CMU LOGIOS Lexicon Tool [5] to generate ARPAbet-transcribed pronunciations of all possible numbers under 10,000 and append these number-to-pronunciation mappings to the updated lexicon.

Numeral	Type/Rule	ARPAbet Pronunciation
a	Special Word	AH
and	Special Word	AH N D
hundred	Special Word	HH AH N D R AH D
thousand	Special Word	TH AW Z AH N D
1	Basic Number	W AH N
5	Basic Number	F AY V
15	Basic Number	FIH F T IY N
20	Basic Number	T W EH N T IY
25	"20" + "5"	T W EH N T IY F AY V
70	Basic Number	S EH V AH N T IY
75	"70" + "5"	S EH V AH N T IY F AY V
100	"a"/"1" + "hundred"	AH/W AH N HH AH N D R AH D
125	"100" + "and" + "25"	AH/W AH N HH AH N D R AH D AH N D T W EH N T IY F AY V
175	"100" + "and" + "75"	AH/W AH N HH AH N D R AH D AH N D S EH V AH N T IY F AY V
1,000	"a"/"1" + "thousand"	AH/W AH N TH AW Z AH N D
1,175	"1,000" + "175"	AH/W AH N TH AW Z AH N D AH/W AH N HH AH N D R AH D AH N D S EH V AH N T IY F AY V

Table 2. Some examples showing the dynamic programming nature of following rules to create normal pronunciation mappings for increasing numbers using previous numbers as lookup table.

Here the threshold of 10,000 is a hyper-parameter,

[4] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[5] http://www.speech.cs.cmu.edu/tools/lextool.html

which is subject to change, since there may be a infinite amount of numbers. Note that the resulting lexicon contains both numeral pronunciation categories, since both are valid ways of pronouncing numbers.

In summary, I create two versions of lexicon. Both versions contain the capitalized word tokens and numerals (both categories). Version 1 attaches punctuations to the end of each possible word token, while Version 2 treats punctuations as standalone word tokens. Thus, for punctuation, the two versions are mutually exclusive.

4.2. Acoustic Model (phone-based)

I follow the acoustic model in the Kaldi Ted-Lium recipe [15, 10] to incorporate phonetic features for punctuation, capitalization and numeral.

1. **Statistical Acoustic Model - GMM-HMM**: Specifically, I first compute the Mel Frequency Cepstral Coefficients (MFCC) of the speech audios using the default 25-millisecond window length and 10-millisecond window step. I then compute their Cepstral mean and variance normalization (CMVN). These data are then fed through monophone (context-independent) and then triphone (context-dependent) training to form the standard statistical GMM-HMM acoustic model. The triphone training includes delta-based, delta + delta, Linear Discriminant Analysis Maximum Likelihood Linear Transformation (LDA-MLLT) and Speaker Adapted Training (SAT) in order. Each monophone and triphone training phase is immediately followed by phoneme alignment to ensure training quality.

2. **Neural Acoustic Model - TDNN**: I then enhance the GMM-HMM acoustic model through Time-Delay Neural Network (TDNN). In this process, I first compute the i-vectors to augment the original MFCC features into 220-dimensional input vectors, align the Feature space Maximum Likelihood Linear Regression (FMLLR) lattices, build a new topology tree before passing the 220-dimensional input vectors into TDNN. Note that it is during this step where I input the pre-trained Ted-Lium model [10] for fine-tuning.

4.3. Language Model (text-based)

1. **Statistical Language Model - 4-Gram**: Using the updated **lexicon** mentioned above, I first train an 4-Gram (ARPA-format) model to create a Lexicon Finite-State Transducer (FST) and a Grammar FST to form a statistical language model. Note that the 4-Gram model also includes uni-grams, bi-grams and tri-grams. I then enhance it through neural language modeling.

2. **Neural Language Model - RNN**: As mentioned in Section 2, the author in [9] discovers that a Transformer-based (e.g.: XLNet) language model in ASR could yield less optimal results (reflected by WER in Table 1) than the conventional RNN language model for small datasets. Because of this, I decide to use RNN instead of Transformer as the neural language model to enhance the statistical 4-Gram language model.

I obtain the final model using beam-search decoding and pruned lattice-rescoring. This is the same as the Kaldi Ted-Lium recipe [15, 10].

4.4. Transcript (data) and Fine-Tuning

I use the first dataset (Ted-Lium) in Section 3 to pre-train a "Traditional ASR" model since Ted-Lium transcripts do not contain punctuation, capitalization or numeral. I then use the second dataset (NSC) in Section 3 to fine-tune this pre-trained Ted-Lium model since NSC transcripts do provide the three aforementioned sentence components. Specifically, I import the pre-trained Ted-Lium model in the TDNN training phase in acoustic modeling of the "Enhanced ASR" model to achieve transfer learning from the "Traditional ASR" model.

Lastly, I use Kaldi's lattice rescoring [15] to compute Word Error Rate (WER) as the evaluation metric and Kaldi's Ted-Lium recipe [10] result as the baseline.

5. EXPERIMENTS AND RESULTS

I first train an "Enhanced ASR" model for each lexicon version using statistical acoustic and language models (mentioned above in Section 4.2.1 and 4.3.1) only to get a preliminary performance comparison between the two versions. Note that this model checkpoint does not take in the pre-trained Ted-Lium model [10] for fine-tuning since fine-tuning takes place in TDNN training for neural acoustic modeling. The middle column of Table 3 shows the WER's of these two model versions using statistical modeling only, where Version 2 clearly outperforms Version 1 in WER on test data (32.5% vs 39.7%), which meets my expectation earlier. The reason may be two-fold: 1). Version 2 has a significantly smaller lexicon size since it appends each individual punctuation as a unique standalone token, where Version 1 attaches each punctuation to the end of each valid word token, thus making it far more impractical. 2). By making each punctuation a standalone token in Version 2, it creates a more natural language model where attaching punctuations to the end of word tokens in Version 1 may incur far more noise in language modeling. This probably better distinguishes punctuations from true silences and pauses.

Then, I choose Version 2 to continue training using neural acoustic and language modeling to enhance its performance (mentioned above in Section 4.2.2 and 4.3.2). Due to time

ASR Model	AM: GMM-HMM LM: 4-Gram	+ AM: TDNN (1 Epoch)
Ted-Lium (Traditional ASR)	*16.1%*	*6.7%*
Version 1 - Punct Attached (Enhanced ASR)	39.7%	-
Version 2 - Punct Tokenized (Enhanced ASR)	**32.5%**	19.9%

Table 3. Word Error Rates (WER) of statistical GMM-HMM + 4-Gram model and TDNN model (I only trained 1 epoch for TDNN and did not run through RNNLM due to time constraint) on test data for the two versions of the "Enhanced ASR" models presented in this paper. Note that I do not run Version 1 "Enhanced ASR" model through neural modeling (TDNN for acoustic modeling and RNN for language modeling) because its performance is significantly worse than Version 2 "Enhanced ASR" model for statistical modeling (GMM-HMM for acoustic modeling and 4-Gram for language modeling). I also attach Ted-Lium results as baseline for comparison, though the last column for Ted-Lium is actually achieved from TDNN and RNNLM. Also note that Ted-Lium result does not include punctuation, capitalization or numeral.

constraint, I only got it to run through TDNN training (for 1 epoch), not through RNNLM. Note that this model checkpoint does take in the pre-trained Ted-Lium model [10] for fine-tuning in TDNN training. The last column of Table 3 shows the WER of Version 2 of the "Enhanced ASR" using TDNN as the acoustic model . By comparing the last column to the middle column for Version 2, we can see that applying neural modeling significantly improves the performance from statistical modeling alone (19.9% vs 32.5%). This is expected as neural modeling should outperform statistical modeling.

In future work, I want to train TDNN for more epochs and actually run through RNNLM to see the limit of this model.

Additionally, I also include the WER's of Ted-Lium on test data [10] as the baseline in the first row of Table 3, for both statistical (middle column) and neural modeling (last column). For statistical modeling (middle column), we can see that even though Version 2 outperforms Version 1, its WER still doubles that of Ted-Lium. Similarly, for neural modeling (last column), we can see that the "Enhanced ASR" (Version 2) also has a significantly higher WER than Ted-Lium. This is also expected, as adding the three aforementioned sentence components definitely incurs additional noise and obstruction to the acoustic and language model. However, the gap between Version 2 and Ted-Lium model is actually decreasing from statistical modeling (middle column) to neural modeling (last column), which is encouraging.

Additionally, I take the liberty to compare the actual decoded prediction result by the statistical and the neural models of Version 2. Table 4 shows an original transcript and the prediction results by the statistical and the neural models of Version 2 on the same speech file. Note that the original transcript is tokenized so that each punctuation is a standalone token for Version 2 comparison. We can see that both versions are able to capture capitalization (other examples can also show the same for numeral), on top of decoding plain word tokens. This asserts the effectiveness of the proposed approach in this work. For punctuation, we can see that the prediction by the neural model is closer to the original transcript than that by the statistical model regarding the middle comma, even though both predictions incorrectly predicts the last punctuation as question mark. This shows that the neural model may be superior in punctuation prediction than the statistical model, which is also expected. This also encourages more work to see the limit on punctuation prediction by TDNN and RNNLM.

ASR Model	Prediction
Original Transcript (Punct Tokenized)	Richard Tay Tian Hoe , and Kok Heng Leun .
Version 2 GMM-HMM	Richard Tay Tian Hoe and Kok Heng Lim ?
Version 2 TDNN	Richard Tay Tian Hoe , and Kok Heng Lim ?

Table 4. A sample prediction by the statistical GMM-HMM Version 2 model and the neural TDNN Version 2 model, compared to the original transcript. Note that the original transcript is tokenized so that each punctuation is a standalone token for Version 2 comparison.

6. REFLECTION

The main bottleneck in this work lies in lexicon construction phase as described in Section 4.1. It is fundamentally difficult to use a brute force approach to cover all possible variants of punctuations, capitalizations and numerals; and it is very much the case that I have not considered all cases in Section 4.1. For instance, I have only considered punctuations that attach to the preceding words like comma and period or as standalone tokens; however, some punctuations may reside inside word tokens like dash. Additionally, some punctuation may have pronunciations other than silence, like "&" which should be pronounced as "and". Also, I have only considered capitalizing the first letter as capitalization, but sometimes the entire word or a portion of the word may be capitalized, such as "ARPAbet" or "US". Moreover, there could be many different ways to read a number (e.g.: year, telephone number, etc.); for example, for a four-digit number like "2589", the current lexicon includes its pronunciation for "two thousand five hundred and eighty-nine", but it is also possible to pro-

nounce it as "twenty-five hundred and eighty-nine", albeit less common. The edge cases above are not included in the current lexicon setup, which may contribute to the relatively high WER by this "Enhanced ASR" model. In essence, to tackle these problems is equivalent to covering all variants of oral expressions in practical scenarios, especially the edge cases, which definitely requires more work.

7. CONCLUSION

I present an "Enhanced ASR" model that includes punctuation, capitalization and numeral in ASR transcript output by modifying the lexicon, acoustic modeling and language modeling components. The key lies in the lexicon pronunciation construction for these three sentence components in addition to plain word tokens. I use statistical learning and neural network approaches in acoustic and language modeling as well as the NSC dataset for transfer learning (from the model trained on Ted-Lium dataset). I compare two versions of the "Enhanced ASR" model with one attaching punctuations to the end of the preceding tokens and the other treating punctuations as standalone tokens and show that the second version outperforms the first version at the completion of statistical modeling stage. Furthermore, I train the second version through neural modeling and achieve better results than the statistical model. However, both statistical and neural models yield less optimal performance compared to Ted-Lium model.

There are three main contributions of this work: 1. It jointly learns and predicts all three of the sentence components in one ASR model, rather than separately, asserted by the evaluation metric and prediction samples. 2. It is an end-to-end model in that it does not introduce additional ASR components or require any middle-ware components to separately deal with each of the three sentence components. 3. It utilizes a transfer learning approach to impart wisdom from SOTA ASR models for plain word tokens to predicting both the plain word tokens and the three aforementioned sentence components.

8. FUTURE WORK

There still remains space for future work. First, I would like to complete a longer TDNN and RNNLM training to see the limit of neural modeling on this particular dataset and language dictionary. Another idea is to conduct ablation studies on the individual and combinatorial impacts of each of the three sentence components. Also, as mentioned above, another idea is to think about efficient ways to consider all possible word capitalizations (a portion of the word may be capitalized rather than the first letter), potential punctuations (a punctuation may reside inside a word token rather than in the end) and number-word combinations (such as "8th", "1st"), without using a fundamentally brute-force approach.

9. REFERENCES

[1] Ottokar Tilk and Tanel Alumäe, "Bidirectional recurrent neural network with attention mechanism for punctuation restoration," in *INTERSPEECH*, 2016.

[2] B. Nguyen, V. H. Nguyen, Hien Nguyen, Pham Ngoc Phuong, The-Loc Nguyen, Quoc Truong Do, and L. C. Mai, "Fast and accurate capitalization and punctuation for automatic speech recognition using transformer and chunk merging," *2019 22nd Conference of the Oriental COCOSDA International Committee for the Coordination and Standardisation of Speech Databases and Assessment Techniques (O-COCOSDA)*, pp. 1–5, 2019.

[3] Julian Chen, "Speech recognition with automatic punctuation.," 01 1999.

[4] Piotr Zelasko, Piotr Szymanski, Jan Mizgajski, Adrian Szymczak, Yishay Carmiel, and Najim Dehak, "Punctuation prediction model for conversational speech," *CoRR*, vol. abs/1807.00543, 2018.

[5] Ottokar Tilk and Tanel Alumäe, "Lstm for punctuation restoration in speech transcripts," 01 2015.

[6] Fernando Batista, D. Caseiro, N. Mamede, and I. Trancoso, "Recovering punctuation marks for automatic speech recognition," in *INTERSPEECH*, 2007.

[7] Eunah Cho, Jan Niehues, and Alex Waibel, "Nmt-based segmentation and punctuation insertion for real-time spoken language translation," 08 2017, pp. 2645–2649.

[8] Eunah Cho, J. Niehues, and Alexander H. Waibel, "Segmentation and punctuation prediction in speech language translation using a monolingual translation system," in *IWSLT*, 2012.

[9] Kareem Nassar, "Transformer-based language modeling and decoding for conversational speech recognition," *CoRR*, vol. abs/2001.01140, 2020.

[10] François Hernandez, Vincent Nguyen, Sahar Ghannay, Natalia A. Tomashenko, and Yannick Estève, "TED-LIUM 3: twice as much data and corpus repartition for experiments on speaker adaptation," *CoRR*, vol. abs/1805.04699, 2018.

[11] Yu-An Chung and James R. Glass, "Speech2vec: A sequence-to-sequence framework for learning word embeddings from speech," *CoRR*, vol. abs/1803.08976, 2018.

[12] Nicola Ueffing, M. Bisani, and Paul Vozila, "Improved models for automatic punctuation prediction for spoken and written text," in *INTERSPEECH*, 2013.

[13] Wei Lu and Hwee Ng, "Better punctuation prediction with dynamic conditional random fields.," 01 2010, pp. 177–186.

[14] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, San Francisco, CA, USA, 2001, ICML '01, pp. 282–289, Morgan Kaufmann Publishers Inc.

[15] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. Dec. 2011, IEEE Signal Processing Society, IEEE Catalog No.: CFP11SRW-USB.

[16] Jia Xin Koh, Aqilah Mislan, Kevin Khoo, Brian Ang, Wilson Ang, Charmaine Ng, and Ying-Ying Tan, "Building the Singapore English National Speech Corpus," in *Proc. Interspeech 2019*, 2019, pp. 321–325.