LANGUAGE MODEL CAPITALIZATION

Françoise Beaufays, Brian Strope

Google

ABSTRACT

In many speech recognition systems, capitalization is not an inherent component of the language model: training corpora are down cased, and counts are accumulated for sequences of lower-cased words. This level of modeling is sufficient for automating voice commands or otherwise enabling users to communicate with a machine, but when the recognized speech is intended to be read by a person, such as in email dictation or even some web search applications, the lack of capitalization of the user's input can add an extra cognitive load on the reader. For these cases, speech recognition systems often post-process the recognized text to restore capitalization.

We propose folding capitalization directly in the recognition language model. Instead of post-processing, we take the approach that language should be represented in all its richness, with capitalization, diacritics, and other special symbols. With that perspective, we describe a strategy to handle poorly capitalized or uncapitalized training corpora for language modeling. The resulting recognition system retains the accuracy/latency/memory tradeoff of our uncapitalized production recognizer, while providing properly cased outputs.

Index Terms— Capitalization, language modeling, FST.

1. INTRODUCTION

In most European languages, casing patterns are part of regular spelling and stylistic rules. The first word of a sentence is typically capitalized, and so are proper names. Acronyms are often fully upper cased (e.g. "IBM", "NASA") whereas abbreviations may vary in their casing (e.g. "CA" or "Calif." for California). Most often capitalization affects a single letter, but occasionally two, as with the Dutch "ij" pattern. Capitalization may be used to various extent in titles ("The Turkish March" in English, but "La Marche turque" in French). In some languages it helps distinguishing parts of speech, e.g. nouns from verbs, adjectives, etc in German ("Arbeit" vs "arbeiten"), in others it is used only for some specific personal pronouns ("arrivederLa" but "arrivederci" in Italian, "I" but "you" in English). And of course fancy mixed cases are routinely chosen for product names ("iPhone", "LaTeX", ...) adding to their visual saliency. Capitalization rules and practices may vary across languages, but it seems fair to claim that for someone familiar with a given language, capitalization is part of the regular patterns we expect, which help make reading faster and more pleasant.

Speech recognition systems however have traditionally ignored capitalization, mostly out of concern that allowing different casings will increase the vocabulary size (e.g. common name "mark" vs proper name "Mark"), which in turn will cause data fragmentation in the language model and erode recognition accuracy. A French study [1] for example describes the out-of-vocabulary (OOV) decreases achieved through text normalization, including lower-casing, on a mid-size (by current standards) language model corpus. In many cases, this simplifying assumption is an acceptable engineering shortcut. With voice search from Google, simple voice actions are legible even without capitalization, and voice queries are directed to a search backend that is case insensitive. However, if the spoken query is longer, and the user wants to glance at the recognition result, or if the application provides SMS dictation or voicemail transcription, intended for a human to read, then capitalization is highly desirable.

To reconcile readability with perceived engineering constraints, capitalization is typically implemented as a post-processing step after recognition, often under the form of a second-pass rescoring. In such implementations, a large capitalized language model is used to restore capitalization, and possibly punctuation, to the first-pass recognition result. Specific implementations include using a simple word mapping as implemented in the SRILM [2] disambig tool, in which case the language model is used for statistical disambiguation, or an FST framework where a lattice of capitalization alternates is generated, composed with the language model, and searched for the most probable rendition [4]. More sophisticated discriminative techniques based on maximum entropy Markov models have also been proposed [8]. A thorough comparison of such methods on Portuguese Broadcast News can be found in [3]. Such approaches however add to the complexity of the overall recognition system, increase its memory footprint, and affects the fluidity of streaming recognition. Moreover, one can easily argue that any statistical knowledge that is relevant to text formatting pertains to language modeling and should therefore be folded in the recognition model.

Following the general trend in the field, the Google recognizer was providing uncapitalized recognition results until recently. The unsupervised speech logs that form the bulk of the model training material were therefore lower case. The rest of the training data, mostly web typed entries, are also not reliable in their capitalization formatting.

In the rest of this paper, we describe how we assembled a corpus of well-formatted text data from which we could learn proper capitalization patterns, and how we included this knowledge in our recognition language model. We characterize the resulting capitalized system in terms of recognition and capitalization accuracy, and provide some error analysis.

Although punctuation could to a large extent be modeled using the same techniques that are described here for capitalization, it is not the object of this paper and will not be described here.

2. INTRINSIC VS. POSITIONAL CAPITALIZATION

As suggested in the introduction, there are, broadly speaking, two types of capitalization; we will refer to them as "positional capitalization" and "intrinsic capitalization". By "positional capitalization" we mean the practice of upper casing the first letter of the first word of a sentence, irrespective of the part of speech or nature of the word. By "intrinsic capitalization" instead, we mean upper casing certain

letters in a word or group of words independently of their position in the sentence (proper names, abbreviations, etc).

As a matter of system design, we would argue that positional capitalization is best handled by the client code running on the final device that renders text to the user, whether typed or spoken. The client has knowledge of the full message being composed on the device independently of how it was assembled, which could include a mix of modalities and potential corrections. Such context information can be communicated to the server-side recognizer, but it seems practical to avoid doing so if not truly necessary. Depending on the application, positional capitalization may be required (for example in a dictation app) or not (there is little need to capitalize the web search 'pictures of kittens'). On the other hand, intrinsic capitalization is best handled on the server side since it is a more complex problem, and requires a larger statistical model to make the right capitalization decisions.

For these reasons, we will leave positional capitalization to the client, and focus exclusively on intrinsic capitalization. That simplification also reduces some of the complexities and potential redundancies for modeling the first word after a start-of-sentence symbol.

3. CAPITALIZATION FRAMEWORK

3.1. High-Level Approach

To train a capitalized recognition language model, we need training data sources that themselves are capitalized. However, our most relevant training data, such as search typed logs or logs from previous spoken interactions with our systems are poorly capitalized when at all. Better formatted data sources such as news corpora, books, or online publications, would offer the right capitalization patterns, but would almost necessarily be mismatched with the tasks users expect to perform by speech, and the language models built from this data would offer lower recognition accuracy than well-matched, but uncapitalized, training data.

For this reason, we adopted a two-step procedure: we first used well-formatted, but somewhat mismatched text to capitalize our language model training data; and then trained final models from these well-matched, and now re-capitalized sources. This way, formatting the data, and matching the data to the domain are essentially kept independent of each other.

We will now describe how to capitalize language model training corpora given a well-formatted, capitalized, language model. After that, we will explain how we trained this reformatting language model.

3.2. FST-Based Capitalization Methodology

Assume a large, capitalized, language model is trained from well-formatted text. We will call this model the Capitalization LM. The Capitalization LM may be trained from a variety of written material. It is expected to have broad coverage, but does not need to be extremely well matched with the target recognition tasks. For reference, the Capitalization LM used in this paper makes on average 25% relative more word errors than the baseline production language model, so it is considered non-competitive as a production model.

The Capitalization LM however can be used to restore capitalization in the training data used to estimate the production model. We use an finite state transducer (FST) infrastructure similar to the one described in [4]. The symbol table of the Capitalization LM is used to derive a mapping from all possible mixed-case words in the

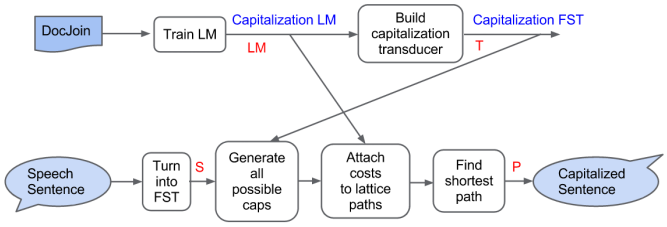

Fig. 1. Capitalization pipeline.

LM to their lower-case equivalent, and vice-versa; e.g. we can derive that "iphone" is rendered as "iPhone" in the Capitalization LM, or that "curiosity" can be rendered in lower case (common noun) or upper case (the Mars rover). This mapping takes the form of a transducer, T, also refered to as Capitalization FST. Counts for the realizations with different cases are not kept.

A lower-cased input sentence to be capitalized is first represented as an FST, S. That FST is composed with T. The resulting lattice contains all the possible capitalization alternatives for the input sentence. The lattice is then composed with the Capitalization LM to apply weights to the various lattice paths. The shortest path, P, through the lattice provides the highest-likelihood capitalization for the given input sentence. This is summarized in Eq. 1 and illustrated in Fig. 1.

$$P = \text{bestpath}(S \circ T \circ LM) \tag{1}$$

Note that this FST manipulation only affects the capitalization of the input text, it does not modify the actual word sequences. This is why the Capitalization LM does not need to be perfectly matched to the target recognition tasks: it just needs to have sufficient coverage and to be close enough to provide the right capitalization pattern for a word, given its context. As another consequence, recognition language models trained from uncapitalized or capitalized data will remain fairly similar.

Another path to accomplish the same result might have been to re-recognize the recognition model training data with the Capitalization LM but, as indicated above, this language model is not very accurate. Moreover, this procedure would be slower than rescoring, and more fundamentally not all the data sources that need capitalization contain audio (e.g. web search logs).

There are two side issues however that arise with the FST approach described in Eq. 1: OOVs and start of sentences, which we will describe next.

3.2.1. Dealing with OOVs

Since the proposed capitalization pipeline relies on a chain of FST compositions whose symbol tables need to match, it is constrained by the vocabulary of the Capitalization LM: a word in an input sentence that is not in the LM vocabulary will break the pipeline. This event can be made arbitrarily infrequent by increasing the coverage of the LM corpus, but some solution needs to be implemented for the remaining cases.

One solution might be to collect all the words in the recognition training corpora ahead of time, and to add them to the Capitalization LM vocabulary list prior to training. This however introduces dependencies that break the nice separation between the upper and lower pipelines in Fig. 1.

Another simple solution consists in checking if any word of an input sentence is out-of-vocabulary for the Capitalization LM, and if so replace it with a rho matcher to pass it unchanged to the output string, or to replace it with the unknown "<unk>" word, or even to

simply delete it from the input string and insert it back in after capitalization (remember, the words don't change, nor their positions). In all cases, the effect will be to leave that word uncapitalized, which is about as well as we can do short of venturing into more complicated solutions including for example semantic parsers.

3.2.2. Dealing with Starts of Sentences

Start of sentences are a more delicate issue. The Capitalization LM is trained from properly capitalized sentences, whose first word is thus capitalized. Likewise, the capitalized training sentences for the recognition language model will be capitalized at start of sentences. Such capitalized words will contribute to n-grams with capitalization patterns that are determined by positional capitalization, not intrinsic capitalization. In turn, these n-grams will cause erroneous capitalization insertions inside sentences when used during recognition.

Conversely, if we decide to remove any positional capitalization and force words at beginning of sentences to be lower cased, we will introduce deletions of needed capital letters, e.g. for proper names inside sentences. Moreover, if the client application doesn't do any positional capitalization, we might see a search query be recognized as "paris France" instead of "Paris France".

A simple solution to eliminate what amounts to positional noise from our data consists of inserting some generic word at the beginning of the sentence to capitalize, and remove it after capitalization. This word will be erroneously capitalized, but inasmuch as it does not break the flow of the sentence and of the LM rescoring, it will allow the rest of the sentence to be properly capitalized. Experimentally, we found that words such as "and" or simply "<unk>" are appropriate for this task, and result in very similar capitalization performance.

4. CAPITALIZATION LANGUAGE MODEL

4.1. Corpus

The above section described how, given a capitalized corpus, one can train a Capitalization LM, and through some FST manipulations, use this LM to capitalize the recognition model training sentences. The question now is how to obtain such a corpus. Again, it is not key that this corpus be extremely well-matched with the recognition tasks since it is only used for capitalization, but large mismatches would affect the FST rescoring procedure, and any training word not in the capitalization LM won't possibly be capitalized. For this reason, we aimed at a broad, typed, varied corpus, and used Docjoin, a corpus that assembles many written documents avalaible on the web.

Understandably the corpus is very large, and not always well-formatted. We therefore applied a series of heuristics to filter the corpus and retain clean utterances.

4.2. Corpus Cleaning

Various adhoc rules were implementd to filter the Docjoin corpus. Since the corpus is extremely large, we aimed at a low-yield high-precision procedure. Rejecting sentences containing non-ascii characters, too many punctuation marks, too many capitalized characters, or too many character or word repetitions eliminated about 25% of the data, and left a qualitatively much cleaner corpus. Perhaps the most efficient filter though consisted in imposing that training sentences be grammatically correct sentences, i.e. that they started with an upper case, and ended with a punctuation mark such as period, question mark, or exclamation mark. This filter alone rejected another 50% of the corpus while strongly reducing the amount of poorly formatted text. Quantitatively, filtering out non-sentences reduces the capitalization error rate (defined below) by 15% to 30% relative, depending on the test set.

The filtered corpus was then used to train a 20M n-gram capitalization language model.

5. BASELINE RECOGNITION SYSTEM

The speech recognition engine used for server-based Google speech applications is a standard, large-vocabulary, recognizer, with PLP features and a multilayer feedforward acoustic model [5].

The language model is trained using 12 data sources extracted from previous interactions of Google users with the speech recognizer and from a variety of typed data sources including web searches and other typed material. Speech data consists of recognition results, not human transcriptions, and to increase the probability that those results are the correct transcriptions, the data is filtered by recognition confidence score. The data sources we used vary in size, from a few million to a few billion sentences, making a total of 7 billion sentences. Individual language models are trained for each data source. These individual models have a maximum of 2M words, n-grams up to order 5, and they are entropy-pruned to have less than 20M total n-grams. Then the individual language models are Bayes-interpolated [6] using a transcribed development set that is representative of the overall recognition traffic. Finally the resulting language model is entropy-pruned down to a size that fits our production targets for memory and latency. The final language model contains a little under 25M n-grams, and the FST representation of the LM occupies roughly 0.5GB. In this pipeline, capitalization is an optional pre-processing step to the training of the 12 individual source language models.

The recognition search is a Viterbi beam-search based on FSTs [7].

5.1. Error Metrics

Capitalization accuracy can be measured in different ways. In particular, it can be included in the word error rate, in which case recognizing "tokyo" for "Tokyo" would be counted as a word substitution error, or it can be assessed separately, in which case "tokyo" would only count as a capitalization error, not a recognition one. To better control the effect of various modeling decisions on capitalization performance, we preferred to keep the two metrics separate. Specifically, we still measure the word error rate (WER) by comparing down-cased recognition and transctiption word strings. Capitalization error rate (CER) instead is computed at the character level. We remove any character that is not upper case from the recognition hypothesis and human transcriptions, and then compare the remaining character strings with the same dynamic-programming alignment used for WER computation. For example, if the recognition result is "Hi Bob" and the human transcription "High top", we compare the strings "H B" and "H", and count one capitalization insertion error for "B". This error metric has the advantage of being very intuitive but it is quite unforgiviging: recognizing "nasa" for "NASA" results in four errors, for a single word.

6. EXPERIMENTAL RESULTS

6.1. Capitalized Language Model Characteristics

We trained a capitalized recognition language model as described in the previous sections, and compared it to a language model trained

Test Set	WER (del/ins/sub) in %	CER (del/ins/sub) in %
Language model with capitalization		
SEARCH	19.3 (3.8/3.8/11.7)	35.7 (27.2/4.7/3.7)
MAIL	8.1 (1.3/1.2/5.6)	26.5 (15.3/9.0/2.2)
ALL	12.5 (2.2/2.1/8.1)	33.0 (19.9/10.7/2.4
Language model without capitalization		
SEARCH	19.4 (3.8/3.8/11.8)	94.4 (93.3/0.5/0.5)
MAIL	8.2 (1.3/1.3/5.6)	56.4 (54.0/1.3/1.0)
ALL	12.5 (2.2/2.1/8.2)	79.1 (76.6/2.0/0.5)

Table 1. Word Error Rate and Capitalization Error Rate with 2 comparable language models, with and without capitalization.

without the capitalization step. Since the language modeling training pipeline targets a vocabulary size and model size that are compatible with our production requirements, the two models have comparable context and total numbers of parameters. Also, the training data sources are intrinsically quite noisy (lots of typos), so OOV rates are not obvious product-impacting metrics either. The most relevant metric to compare the two language models remains recognition accuracy, in addition to capitalization of course.

6.2. Recognition and Capitalization Accuracy

Table 1 below reports the accuracies of the capitalized and uncapitalized models on three respresentative data sets: a search by voice test set, a Gmail dictation test set, and a generic test set whose distribution reflects that of all of our current incoming speech traffic. The test sets vary from 15K (Mail) to 45K (Search) utterances, and from 100K to 200K words each. All three were carefully transcribed, and transcribers were instructed to use capitalization according to American English usage.

As can be seen from Table 1, recognition performance is roughly identical whether the language model is capitalized or not. At least for large vocabulary systems, the fear of increasing the vocabulary size and fragmenting the training data was thus unfounded. One would actually hope that allowing multiple pronounciations for words that only differ by their casing, e.g. "US" vs "us", would make the capitalized language model more accurate. So far, we added these subtleties sparingly, and perhaps the very slight differences between the two systems can be attributed to this slight lexical refinement.

Capitalization performance is good, but there's a lot left to improve. The capitalization error rate (CER) is dominated by deletions, and we will provide some analysis of these errors below. It should be noted first however that the baseline, uncapitalized, system contains some basic mechanism to correct the most embarassingly obvious missing capitals, such as in the first person personal pronoun. This explains the less than 100% capitalization deletion rate in the lower portion of the table for the baseline system.

6.3. Error Analysis

A qualitative look at capitalization errors quickly hints at consequences of recognition errors. This correlates with the observation that capitalization errors are dominated by deletions: randomly replacing a word by another is more likely to replace a capitalized word by an uncapitalized one than vice-versa. Quantitatively, sentences in the generic "ALL" test set that have at least one misrecognized word have a 20% (absolute) higher capitalization error rate than correctly recognized sentences. And capitalization errors from utterances with a recognition error account for 65% of the total number of capitalization errors.

More interesting perhaps are the capitalization errors made in otherwise correctly recognized sentences. On a random sample of error sentences that were manually labeled, the largest single category of errors related to addresses and businesses (e.g. "Rancho Valencia resort", "Salvation pizza Houston") which accounts for 25% of all capitalization errors. This points at a lack of coverage of such entities in the capitalization corpus. Another 13% of the errorful sentences contained song or movie titles and names of music bands (e.g. "Silversun Pickups kissing families"). Another 31% contained random errors, including capitalization deletions and substitutions. Finally 31% were either ambiguous cases or human transcription errors.

Our assumption at this point is that a large fraction of the observed errors could be corrected by increasing the coverage of the capitalization corpus to include more Maps and YouTube entities to cover more locations and titles. Fewer errors relate to more subtle distinctions such as the common "mom" vs "Mom" confusion.

7. CONCLUSION

In this paper, we explored the hypothesis that a large-vocabulary, production-quality, language model can be trained to encode capitalization in its constituent n-grams. We showed that the tasked-matched data used to train the recognition model can be capitalized using case patterns learned from a well-formatted data corpus that was not well matched to the target recognition tasks, thereby separating formatting and accuracy into two independent problems. Our experiments confirm that the capitalized recognition system maintains the accuracy and resource tradeoffs of the uncapitalized system, while providing fairly accurate capitalization.

The methodology proposed in this paper can easily be generalized to diacritic restoration, spelling correction, and to some extent punctuation rendering.

8. REFERENCES

[1] G. Adda, M. Adda-Decker, J-L. Gauvin, L. Lamel, "Text normlization and Soeech Recognition in French", Eurospeech 1997.

[2] SRILM, http://www.speech.sri.com/projects/srilm

[3] F. Batista, D. Caseiro, N. Mamede, I. Trancoso, "Recovering Capitalization and Punctuation Marks for Automatic Speech RecognitionL Case Study for Portuguese Broadcast News", Speech Communication, Vol 50, Issue 10, Oct. 2008, pages 847-862.

[4] A. Gravano, M. Jansche, M. Bacchiani, "Restoring Punctuation and Capitalization in Transcribed Speech", ICASSP 2009.

[8] C. Chelba and A. Acero, "Adaptation of Maximum Entropy Capitalizer: Little Data Can Help a Lot", EMNLP 2004.

[5] N. Jaitly, P. Nguyen, A. Senior, V. Vanhoucke, "Application of Pretrained Deep Neural Networks to Large Vocabulary Speech Recognition", Interspeech 2012.

[6] C. Allauzen, M. Riley, "Bayesian Language Model Interpolation for Mobile Speech Input", Interspeech 2011.

[7] OpenFst Library, http://www.openfst.org