

University of Lorraine | IDMC | LORIA

MSc Natural Language Processing – 2018 - 2019
UE 805 – Supervised Project

Transcribing Out-of-Vocabulary words

Students:
Elena Khasanova

Supervisors:
Denis Jouvet
David Langlois
Reviewer:
Christophe Cerisara

May 29, 2019

Contents

Introduction

Automatic speech recognition systems (ASR) convert spoken utterances into a strings of text to enable further processing of the input, for instance, semantic or syntactic parsing, key-word search, document retrieval, etc. These systems commonly use reference vocabularies containing words and their phonemic representations. Regardless of the size of the vocabulary, there are always words that can not be accounted for, so called out-of-vocabulary words (OOV), which not only leave parts of the input unrecognized but also confuse the surrounding context thus constituting a problem for speech recognition systems. An efficient transcription systems should potentially be able to handle any possible input with a known alphabet. There exist multiple approaches to recognition and speech-to-text conversion of out of vocabulary words performing with different efficiency.

The current project aims to review the state-of-the-art in the field and experiment with several types of data and conversion methods that offer transcriptions of OOV. The project is divided into two phases: in phase one, we observed and compared the existing conversion methods presented in the literature, the second part builds on the results obtained through the analysis of literature and is devoted to the empirical experimentation with transcribing methods. The project is analytical in nature, the implementation of a novel tool for phoneme-to-grapheme conversion is out of the scope of this project. The present report is a reflection on the second stage of the project, which is implemented towards the fulfillment of the requirements of M1 degree in Natural Language Processing at the University of Lorraine. The project is conducted under the supervision of Denis Jouvet, Senior Researcher at INRIA and the head of Multispeech team at Lorraine Research Laboratory in Computer Science and its Applications (LORIA), and David Langlois, the Assistant professor in Informatics at the University of Lorraine and a member of SmarT team at LORIA.

The following objectives were established for this part of the project:

- to select an approach for phoneme-to-grapheme conversion for exploration and adjustment;
- to select the appropriate data for conversion;
- to define the possible parameters and input types to adjust;
- to conduct experiments with various models, parameters and input types and assess their performance;
- to explore approaches tor transcribing a sequence of phonemes into a sequence of letters and word boundaries for a speech segment containing several words, all unknown or a combination of known and unknown words;
- to decide on the evaluation metrics and develop necessary evaluation tools;
- to compare the results to the baseline model and analytically process them.

Joint-sequence model offered by M. Bisani and H. Ney [1] was selected as a basis for our analysis as it was found reasonably performant, comprising multiple parameters to adjust and alter, and freely accessible. The conversion was done on Carnegie-Mellon University Pronouncing Dictionary using Sequitur Grapheme-to-Phoneme data-driven converter written at RWTH Aachen University by Maximilian Bisani.

To implement this project, Python was used (versions 3.6 and 2.7)[2].

Three metrics are used to determine the quality of conversion: word error rate, character error rate, and recall.

The structure of the report

This report consists of three parts. In part 1, the state of the art in phoneme-to-grapheme conversion is discussed focusing particularly on the selected joint-sequence hybrid model. It is followed by the presentation of the data used in the project. Further, the experiments with various parameters and inputs are introduced. It includes transcribing sequences of phonemes for a single word, transcribing sequences of phonemes for a group of words with a challenging task to determine a word boundary, and finally converting the sequences of phonemes for an unknown input type: a group of words of varying length or a single word. The technical aspects of language modeling are also discussed in this section. The analysis of the results obtained from the experiments from part concludes the report.

Chapter 1

The model

1.1 Joint-sequence-model as state-of-the-art in phoneme-to-grapheme conversion

Automatic phoneme-to-grapheme conversion is considered in the context of speech-to text applications such as voice user interfaces, voice search, voice typing, subtitling, and other. Regardless of a small symbol inventory (alphabets are normally limited to a few dozens characters) and seemingly straightforward pronunciation rules, it is a challenging task due to the high pronunciation variability, from regional or individual pronunciations to the fundamental asymmetry of phoneme-letter correspondences, a large number of differently transcribed homophones, exceptions and irregularities, and context interference. Out-of-vocabulary words impose an extra difficulty as there is no one-to-one correspondence between the phonetic and graphic forms in the majority of languages. It becomes particularly complex for the languages with writing systems that do not unambiguously represent phonology, such as French or English. A number of solutions to tackle this issue have been offered.

In Part 1 of the project, we investigated modern approaches to phoneme-to-grapheme conversion along with the methods to deal with out-of-vocabulary words. They include dictionary lookup, rule-based conversion, classification-based models, prediction by analogy, probabilistic approaches as well as neural networks and joint-sequence models. The common methods to process OOV words include using an unknown word model in the speech recognizer, classifying the OOV, expanding the reference vocabulary with the new words, and learning sub-word units to expand the vocabulary. The most important aspects to consider are the quality of recognition, alignment accuracy, and treatment of OOV words.

Simple to implement approaches such as dictionary lookup require extensive preprocessing and large storage space. Despite being based on absolute matching and thus error-free, they are extremely costly to create and can not handle OOV words efficiently. Rule-based systems provide almost full coverage and minimize error, but they require a costly and careful rule-design procedure based on extensive linguistic knowledge. Classification-based alignment depends heavily on multiple preprocessing steps and is tuned to the post-processing steps and is thus lacking flexibility. Prediction by analogy can perform well only if the training data is selected in a careful and balanced way as it emphasizes the most frequent cases. Probabilistic approaches combining various alignment methods, regression trees, n-gram models and scoring strategies were found the most efficient. Two approaches discussed in the literature stand out: becoming increasingly popular configurations of neural networks and joint-sequence model, which show similar performance.

Research shows that neural networks may bring significant benefits while maintaining high performance rates. They are small in size and advantageous in handling both alignments and recognition and can transcribe data directly from acoustic input avoiding the intermediate

phonemic representation. The best possible results are achieved from the implementation of a combined deep bidirectional long short-term memory neural network with connectionist-temporal classification and 5-gram finite-state transducer model presented in Graves Jaitly [3]. This system uses spectrograms derived from raw audio files as the minimal preprocessing scheme, which are then processed by a DBLSTM network with 5 hidden layers trained on text transcripts. In multiple experiments on WSJ data (14 hours and 81 hours training corpora), the best result was 6.7% WER on test set and WER of 21.3% on the public CMU dataset.

It was concluded that the most promising and beneficial for the goals of this project approach is flat-hybrid joint-sequence model proposed by M. Bisani and H. Ney ([1]), [4], which is also applied in Sequitur converter developed by the same authors.

The model is based on a combination of the pronunciation model and sub-lexical language model into a "graphoneme" units, or "graphone". A "graphoneme", or "graphone", is a pair of a letter sequence and a phoneme sequence of a different length. The set of graphemes and the set of phonemic transcriptions are aligned based on Bayes decision rule: for a given orthographic form, the most likely phonemic representation is sought, which was confirmed optimal with respect to word error rate. The letter and phoneme sequences are co-segmented (grouped) into an equal number of segments, but grouping may not be unique, thus it is called m-to-n alignment. As an example, the pronunciation of the word 'clicking' can be represented as a sequence of graphones in any of the following ways, with the first case being many-to-many alignment and the second 1-to-1 correspondence:

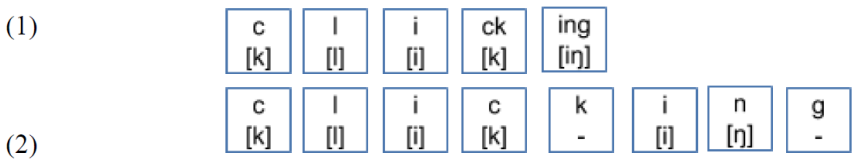

Figure 1.1: Alignment [4]

The graphone inventory may be specified by hand or obtained from training data. The graphones are constrained by two parameters - the length of the graphone inventory and the maximum history length (the number of elements that affect the probability of a graphone at a given position). A joint unit carries both input and output sequences. When the number of input and output symbols are from 0 to 1, the model corresponds to a rule-based finite state transducer, while with more co-sequences it is represented as a joint multigram.

The multigrams are defined based on maximum likelihood training with the expectation maximization algorithm, which iteratively recomputes the parameters for the models from unigram to the selected value of n to obtain the best likelihood of observation. After the model estimation, the Bayes decision rule is applied to phonemically transcribe the words in the test corpus by maximum approximation: the most likely graphone is selected and projected onto the phonemes. Models of joint-sequences of various lengths (from unigrams to 8-grams) were reported as rather efficient [Galescu and Allen, 2002; Vozilla et al, 2003; Caseiro et al., 2002; Chen, 2003 as cited in Bisani Ney [1]].

The probability distribution is measured on this joint unit and modeled using a standard n-gram of varying order.

The algorithm consists of several steps. The model is trained on a pronunciation dictionary, in which the graphic forms are paired with their pronunciations, but without the alignment on the level of letters and phonemes. All the possible co-segmentations and their probabilities are computed using the standard n-gram model. Co-segmentation (cs) uniquely defines the probability of joint sequence according to the formula:

$$p(g, f, cs) = p(q)$$

in which g is a grapheme, f is a phoneme, cs is a hidden variable representing co-segmentation, and q is a pair of graphemes and phonemes.

There are two varying parameters in the segmentation process: the upper limit L for g and f, excluding the case when one of the elements in a graphone is empty, it limits the size of graphone inventory, and the maximum history length, or the size of n-gram.

The probabilities are iteratively estimated with bottom-up expectation maximization algorithm.

Iterative model estimation with expectation maximization [1]

1. Initialize the unigram model.
   The basic case is a context independent unigram with flat probability distribution, the model estimation always starts at this level.

2. Initialize discountinly graphone length constraints

3. Until the likelihood stops increasing:

   Divide training set into two parts: compute evidence values on the bigger part, adjust discount values on the held-out set:

$$e(q, h; \vartheta) := \sum_{i=1}^{N} \sum_{q \in S(g_i, \varphi_i)} p(q | g_i, \varphi_i; \vartheta) n_{q,h}(q)$$

Figure 1.2: [1]

4. Update the discounts if it did not increase

5. Update the model with applying absolute discount with interpolation also known as Kneser-Ney smoothing, according to the formula

6. Increase the length of n-gram, repeat steps 1-4 until the desired n-gram order is reached.

The new graphone is obtained each time there is non-zero probability for it. There is a threshold on the probability estimates, so that the unlikely graphones are removed during iteration. In certain words, however, graphones with low probability can have large conditional probability, these graphones should not be removed from the model, and evidence trimming, another special threshold similar to the one used with the probability estimates, is introduced. Both thresholds are adjusted on the held-out set. All the possible co-segmentations and their probabilities that pass the threshold are computed and stored in the model.

After the model estimation, the conversion can be applied. As the model deals symmetrically with both phonemes and graphemes, it can be applied to both obtaining phonemic transcriptions from the graphic forms and vice-versa. The most probable graphone sequence matching the given spelling / pronunciation is obtained through the maximization of the joint probability over the matching graphone sequences:

$$p(g, \varphi) = \sum_{q \in S(g,\varphi)} p(q)$$

Figure 1.3: [1]

and then projected onto the phonemes / graphemes respectively. If the same transcription generated several times by a different alignment, the probability of this sequence is summed up.

As the history length is finite, the model is represented as a FST, with each history being a state.

Based on various experiments, Bisani and Ney[1] conducted model evaluation and decided the optimal parameters for several constraints. The accuracy estimates used in this process were PER (phoneme error rate) and WER (word error rate). The model was evaluated based on randomly partitioned disjoint training and testing sets derived from several corpora differing in grapheme and phoneme inventory. The test corpora provide reference words and their phonetic transcription and is presented for English, French, and German languages. The authors conclude that the bigger n-grams yield better performance with a smaller size of graphone. The 8- and 9-n gram (which is an average word length) gave the best result in combination with a singular graphone determining one-to-one alignment. An estimated optimal size of graphone is 4 elements as the larger size causes data sparseness, and smaller size overloads with too many short words with high probabilities. This value for the graphone size constraint is applied in the Sequitur converter used as the basic tool for this project. 1-to-n alignments also show comparable accuracy. The model was reported to efficiently group symbols into larger chunks representing frequent letter groups (e.g. 'th', 'ph') and to deal with OOV words: instead of modeling them intentionally, it constructs any word out of sub-word units, and the transcriptions are consistent with the pronunciations taken over from the training dictionary.

The best performance was obtained for Celex1 data set on 9-gram model, with the WER 2.50+- 0.11 and PER 11.42 +-0.43. The results on CMU dictionary for grapheme-to-phoneme conversion:

Work	CER	WER
Galescu and Allen(2002)	7.0	28.5
Chen(2003)	5.9	24.7
Bisani and Ney(2008)	5.88	24.53

The performance in P2G conversion is slightly worse and language dependent: high accuracy for German is explained by its phonetic orthography, while French with an exceptional number of homophones is strikingly challenging for the system. For the English NETtalk data set the results are almost the same (10% higher letter error rate than phoneme error rate). For CMUdict data the difference is twice larger:

Work	CER	WER
Galescu and Allen(2002)	11.5	49.7
Bisani and Ney(2008)	10:35	47.31

This discrepancy can be explained by the higher number of proper names, abbreviations and acronyms in CMUdict.

Other advantages of the model include easy integration with standard speech recognition architecture as graphones can be added to the pronunciation dictionary. The model is also confirmed efficient in handling the unseen data (OOV) and can be symmetrically applied to G2P and P2G conversion, however, as it memorizes long sequences, it is memory consuming. Another known shortcoming of the model is its inability to determine the word boundaries.

Overall, several tendencies in the evolution of phoneme-to-grapheme conversion systems have been observed: the systemsâĂŹ architectures are becoming more complex as they incorporate several interacting methods and layers, with multiple parameters dependent on one another, the training data fed into the system is also expanded. The systems strive to find one-to-one correspondences between the phonetic and orthographic sequences and avoid ambiguity and perform best when they estimate the model on a context close to the average word length for a particular language.

Graphemes	Phons	Length	Ph/word	Pron/word	W/train	W/test
27	39	7.5	6.3	1.06	873	12000

Grapheme inventory, Phoneme inventory, Average word length , Average phonemes per word, Average number of pronunciations per word,
Orthographically distinct words in training, Orthographically distinct words in test

1.2 Data

Two types of data are used in the present project: the pronunciation dictionary and a text corpus for training a character-based language model.

We use Carnegie-Mellon University Pronouncing Dictionary[5], an open-source machine-readable pronunciation dictionary for North American English. It comprises 134k (including duplicates) words paired with the sequences of phonemes representing their pronunciation. The set of phonemes includes 39 phonemes (mappings to APRAbet set) and stress markers on vowels: 0, 1 and 2 for no stress, primary stress, and secondary stress respectively. APRAbet set represents phonemes and allophones of General American English with unique sequences of ASCII characters. The joint-sequence model was trained on the 90% of the data (around 120400 words), the rest were used as a test set.

Many items contained in the dictionary are given several pronunciation options due to varying stress patterns or letter-phoneme correspondence. For instance, in the following example, the varying parts are stress and the pronunciation of the initial vowel:

ACCENT ÂH0 K S EH1 N T
ACCENT(1)ÂE1 K S EH2 N T

In this set the stress patterns remain the same, while the phonemic pairings vary mainly for vowels and consonants:

ACERO AH0 S EH1 R OW0
ACERO(1) AH0 S Y EH1 R OW0
ACERO(2) AH0 TH EH1 R OW0

The dictionary accounts for the regional pronunciations (for instance, C paired with TH is typical for the speakers of English with Hispanic origin), phonetic reductions, coarticulation.

Many words contain several options of pronunciation, which are labeled with numbers: for example:. The set of phonemes includes 39 phonemes (mappings to APRAbet set) and stress markers on vowels: 0, 1 and 2 for no stress, primary stress, and secondary stress respectively. ÂăThe training data comprises 90% of the original dictionary, which makes 120395 words, the test data includes 13378 words. The words that have several pronouncing options are repeated in the dictionary as each pronunciation variant is counted as a word.

The dictionary, due to its large size, contains a significant number of rare words, proper names and wordforms.

The dictionary is used for training a statistical phoneme-to-grapheme model that can generate transcriptions for the unseen sequences of phonemes.

The CMU pronouncing dictionary is stored as a text file of the following form: one word and its phonemic representation separated with whitespace. Each phoneme is separated from the other by the white space.

Chapter 2

Experiments

2.1 Experiments

2.1.1 Performance metrics

Three metrics are used to evaluate the accuracy of phoneme-to-grapheme conversion: word error rate, character error rate and recall.

Word error rate estimates the discrepancy between the hypothesis (the output of the converter) and the reference (the original word) and shows the number of mismatches in the file divided by the total number of words in the file.

These are not necessarily the distinct words: in case several pronunciation sequences were given initially for the same word or we obtained multiple conversion variants in n-best experiments, each variant was paired with its reference and therefore counted as a word. Word error rate is an absolute measure of accuracy sensitive to near misses: one character mismatch is counted as an error. Character error rate (CER) is measured with Levenshtein distance: the minimum number of single-character edits (insertions, deletions or substitutions) required to change the hypothesis into the reference. The total number of operations required to transform each transcription into the reference divided by the total number of characters is reflected as CER. Recall estimates the number of conversions present in the lexicon. The transcriptions counted as correct are those present in the reference lexicon but they may not match with the original string. Recall shows how well the model can replicate the sequences encountered in the training data, however, it is insensitive to the actual accuracy of the conversion.

2.1.2 Training

F The models for phoneme-to-grapheme conversion are estimated with Sequitur, a trainable grapheme-to-phoneme converter developed at RWTH Aachen University - Department of Computer Science by Maximilian Bisani. It is noted as suitable for training systems with the small size source and target alphabets (less than 255 symbols). The latest release of sequitur (2016-04-25) is used to train the joint-sequence model described in Chapter I on CMUdict (release 0.7b) data.

Two sets of models were trained: models of orders 1 to 7 were trained for on the dictionary in its initial form, with one word and its respective pronunciation per line, and the models of orders 1 to 7 on the combination of words and a word boundary symbol. The model was iteratively trained with the standard graphone inventory limit used in Sequitur, which allows up to 1-to-4 alignments.

The efficiency of the 4, 5 and 7-gram models is shown in the table.

As can be seen, the error rate drops noticeably with the increase of the length of n-gram.

G2P conversion: Sequitur model estimations:

		Word Error Rate	Character Error Rate
4gram	train	24.66	5.60
	test	43.9	11.16
5gram	train	12.83%	2.74%
	test	40.88%	10.25
7gram	train	7.53%	1.62%
	test	39.62%	9.86%

Figure 2.1: [1]

The results reported on a similar data set (CMUdict-06) on 9-grams in literature ensure the decreasing trends. As was reported above, the optimal size of n-gram model is 8 or 9, which corresponds to the average length of a word:

Author	CER	WER
Galescu and Allen (2002)	5.9	24.7
Chen (2003)	7	28.5
Bisani and Ney (2008)	5.88+-0.18	24.53+-0.65

The results for phoneme-to-grapheme conversion are notably worse:

Model	WER	CER
4 gram (test)	49	1.32%
7 gram (test)	4 7	1.27%

Similar results were reported by Bisani Ney: 47.31+-0.73 WER and 10.35+-0.22 CER and Galescu (2002): 49.7 WER, 11.5 CER. It should be noted that, when estimating character error rate, the result was normalized by the number of correct characters instead of the total number of letters in the file. This explains the discrepancy between the CER with similar estimations for WER.

2.1.3 Multiple conversions

The estimations outlined in the previous chapter show the accuracy of the conversion for the best option decided by the converter with all the probability mass put on it. However, given the error rate, the converter might choose not the correct option, especially when deciding between the transcriptions with close probabilities. One of the possible ways to observe this is to compare n-best solutions. The options generated by the system reflect the natural ambiguity and variation found in the training data and are assumed to improve the accuracy of the system.

Several questions arise regarding this:

- does the best option change when multiple conversion options are allowed?
- what is the optimal number of n-best options?
- can incorporating an additional language model improve the decision process?

In order to provide the answers to these questions, we obtained the lists of 5, 10, 20 and 50 best conversions with 4 and 7-gram language models. The output contains the original word, the rank of the transcription, the probability assigned to the transcription by the converter, and the transcription. The example of 3 best conversions list is shown below:

A 0 0.929401 A
A 1 0.037814 A (1)
A 2 0.008425 A ’
AACHEN 0 0.192166 O C O N
AACHEN 1 0.114390 O C K E N
AACHEN 2 0.074778 A C O N
AARDVARKS 0 0.303469 A R D V A R K S
AARDVARKS 1 0.152436 A R D V A R X
AARDVARKS 2 0.055021 A R D V A R C S

2.1.3.1 Recall

First, to understand whether multiple results bring us any benefits in terms of reflecting the original graphic forms, we calculated the recall. The best transcription is decided by searching for the transcription in the lexicon. As the transcriptions are sorted from the most to the least probable, the search continues until the transcription present in the lexicon is matched. For about a half of all the words in the test set, the top conversion is selected (given the estimated p2g error rate). In case the transcription was not found in the lexicon, it was added to âĂIJno conversionâĂİ rate.

The recall measure is obtained by comparing the transcriptions to the words in the lexicon. The comparison was conducted on the test lexicon and the full lexicon. WER was estimated for the conversions that are present in the lexicon but do not match the original strings. CER took into account all the possible conversions.

The results are displayed in the table (the percentage rates are rounded up to the integers).

Multiple conversions indeed seem to be beneficial: the percentage of words not found in the lexicon is significantly smaller than the error rate even for the lower order model and on average is about 10% smaller. Only 3% of transcripts are not present in the lexicon when the estimations are done on the full lexicon and 20 possible conversion options given by 7-gram model.

Several trends can be observed. For the 4-gram model the WER is better when evaluation is done with the test lexicon. The explanation is rather simple: the more transcriptions are

P2G conversion: Recall

		Word error, %				No Conversion, %				Character error, %			
		5 best	10 best	20 best	50 best	5 best	10 best	20 best	50 best	5 best	10 best	20 best	50 best
4 gram	Full vocab	31	26	31	31	11	7	11	11	4.5	5.4	6	6.6
	Test vocab	19	20	20	20	18	18	18	18	4.5	5.4	6	6.6
7 gram	Full vocab	32	32	28	32	9	9	3	9	2.1	2.8	4.2	6
	Test vocab	18	18	9	18	16	16	6	16	2.1	2.8	4.2	6

Figure 2.2: P2G Conversion: recall

recognized as âĂIJnot in the listâĂİ, the fewer transcriptions are counted as mismatches. The total error rate remains almost the same for any number of conversions: around 42 on the full lexicon and around 36-38 on the test lexicon. However, for the 10 best list the rate drops slightly resulting in 33% on the full lexicon, while for 20 options it goes back to the average rate. A value between 10 and 20 options seems to provide the balance between recalling the patterns observed in training and avoiding confusion and sparseness. However, 4-gram model was concluded less efficient by Bisani Ney [1], the higher order models should give better results.

For 7-gram model, a significantly better result is achieved on a list of 20 best conversions on both full and test lexicon with total error rate of 31 and 15% respectively. Other options resulted in the same numbers, which suggests that in most cases the same best match was selected. 5 and 10 best results per word were not enough to select more matching transcriptions, while 50 best options were too many. The test lexicon exactly matches the set on which the conversion was performed thus bringing less confusability to the system.

All in all, using multiple conversions to decide on the best output seems useful to improve the accuracy. In the next part, we will discuss the strategies to improve the decision process.

2.1.4 Character-based language model

Using a character-based language model (CBLM) for selecting the best transcription seems an intuitive solution. CBLMs use sequences of characters from the input data to provide the probability distribution of each character following a sequence of characters of an arbitrary length (history) specified by the order of the model, ranging from unigrams (raw occurrences of characters) to multigrams of unlimited length. CBLMs rely on a very small discreet space (we use only 26 lowercase English letters and several additional characters seen in the training data, such as an apostrophe, underscore, full stop, a dash and an empty string).

Character-based language models are known to be efficient in handling orthography-related tasks, such as suggesting or correcting spelling. We assume that character-based language model can improve the selection of the best candidate output by the joint-sequence model based converter.

There exist multiple techniques to organize and store the models as well as to deal with unseen character sequences.

One of the most commonly used and best performing methods are backoff model and interpolated Kneser-Ney algorithm [6]. According to Chen and Goodman [7], these two models perform the best in many contexts, but most of all when the counts are relatively small. Given that the longer n-gram sequences were These two models were implemented in our project.

Further the implementation details and the experiments we conducted will be discussed.

2.1.4.1 Stupid Backoff

According to Brants et al (2007), a simple backoff algorithm with weighted counts may be sufficient for certain tasks and types of data, especially in case of large language models. âĂIJStupid backoffâĂİ model does not produce a true probability distribution, discounting is applied only in case of zero counts.

If a higher-order n-gram has a zero count, we simply backoff to a lower order n-gram, weighed by a fixed (context-independent) weight. It terminates at unigrams, which have probabilities S(w) = count(w) / N (all counts).

2.1.4.2 Kneser-Ney model

Interpolated Kneser-Ney smoothing algorithm is known to be the best performing n-gram smoothing metihods. It implements a more complex discount formula based on absolute discounting. Discounting of the counts for frequent n-grams allows to save some probability mass for the rare and unseen n-grams. In absolute discounting method, a fixed discount is subtracted from each count. As it is very small, it should not affect the estimations significantly. For smaller counts, the change will be more notable, but small counts are meant not to be trustworthy. Kneser-Ney algorithm takes into account not only the frequency counts, but the number of different contexts a character can appear in as well as the number of characters to follow the history. It estimates the probability of a character to appear as a novel continuation of the history.

The probabilities are estimated recursively according to the formula:

$$P_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\left(c_{KN}(w_{i-n+1}^{i}) - d, 0\right)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda\left(w_{i-n+1}^{i-1}\right) P_{KN}(w_i|w_{i-n+2}^{i-1})$$

Figure 2.3: Kneser-Ney smoothing

There are three components in this formula. The first component shows how many times the selected character appears as a novel continuation out of total number of n-gram types of the highest order. In the numerator, we have either the frequency (= count) of that full n-gram (= c_KN) minus a discount factor called d, or zero, whatever is largest (max). The denominator is the frequency of the history.

For instance, if we deal with a trigram consisting of the characters 'c', 'a', 't', the formula for the first component will be the following:

max (count for 'cat' minus d, 0) / count for 'ca'

In other terms the split is the following:

The second component, lambda, follows its own formula and is usually precomputed for each history and stored in the model. Lambda is independent of the character. To compute lambda coefficient, we divide the number of different characters that follow the history multiplied by the discount value by the frequency of the history in the model.

$$c_{KN}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for the highest order} \\ \text{continuationcount}(\cdot) & \text{for lower orders} \end{cases}$$

Figure 2.4: Continuation count formula

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w : c(w_{i-1}, w) > 0\}|$$

Figure 2.5: Lambda value

In our example, we will compute lambda for 'c a t' in the following way:

lambda = d * number of distinct characters that follow 'ca' / frequency count of history

Finally, the third component is the continuation probability of a character to follow the reduced history, which is computed according to the same initial formula.

In this case, we will compute the probability of 't' to follow 'a. We will iteratively recompute the probabilities until we reach unigram counts. In the unigram case, the raw probability of a character is the frequency of the character in the training corpus divided by the size of the corpus.

This model takes into account the probability of a character in lower order n-grams when computing probabilities in the higher orders.

In case the history is not seen in the corpus, we back off to a lower order. When the needed character is not seen in the history, the first component is disregarded (as it will be equal to 0).

2.1.4.3 Implementation

Two types of standard n-gram language model were implemented: a so-called Stupid Backoff model and a backoff model with Kneser-Ney smoothing. Both models can be trained on any raw text.

Both models are implemented as Python Counter default dictionaries and are based on seen data. The algorithm for constructing the backoff models is presented below:

1. The frequency counts are obtained for each history-character sequence, depending on the n-gram order that indicates the length of the history.

2. For each length of the history n, the model of n-1 order is iteratively estimated and appended to a list of dictionaries so that the first model in the list is always the model with the desired history length, and the last is the unigram model with basic counts for each character following an empty string history.

3. With simple backoff model, the counts are normalized by the number of occurrences of all the possible characters after a given history.

4. With Kneser-Ney smoothing model, instead of the normalized frequency counts reflecting the probability, the actual counts are stored. Also, for each history a smoothing coefficient lambda is computed and stored in the model in a 3-tuple

5. The model is serialized in a pickled object.

The two models differ in the way they estimate the probability of a character to complement the given history.

Backoff model follows this strategy:

1. Search for the character-history pair starting from the model at index 0 in the list of models (the highest order n-gram)

2. If it is not found, decrease the history by removing the first character, decrement the order, increment the index. Repeat steps 1-2 until the needed pair is found, extract its probability from the tuple. If the pair is not found at one-character long history and the index reached the last model, change the history to the empty string and obtain the unigram probability.

3. Each time the model is backed-off to the lower order, apply the discount value. The resulting probability is obtained from the formula:

$$P(ch|hist) = c(ch|hist) \ / \ C(hist) * d **i$$

with c being the frequency count of a character-history pair, C(hist) being the frequency counts of all the characters following the history (normalized value), d being the arbitrary given discount, and i being the index of the model the function backed off to.

Kneser-Ney model

1. Search for the history starting from the model at index 0 in the list of models (the highest order n-gram)

2. If the history is not found, decrease the history by removing the first character, decrement the order, increment the index. Repeat steps 1-2 until the needed history is found, extract the lambda from the tuple.

3. If the history is not found at one-character long history and the index reached the last model, change the history to the empty string and obtain the unigram probability

4. Once the history is found, recursively compute the probability according to the formula of Kneser-Ney smoothing.

2.1.5 One word input

We applied the Stupid Backoff and Kneser-Ney language models to the lists of n-best conversions to see whether there is any observable improvement in the error rates.

The models were trained on the lexicon extracted from CMUdict.07 with one word per line. Special boundary symbols to indicate the start and the end of the word are added to each word. The discount value d used for both models is 0.75 as it was reported in the literature as the most pertinent [7].

We compared the results obtained from the models of different orders with respect to the order of the model applied in the conversion: we applied 3 to 5-gram models to the transcriptions generated by the model of order 4, and 6 to 9 gram models to the transcriptions generated by 7-gram model. The joint-sequence model has not been trained further due to the limitations in time and equipment available during the project.

As the average word length in the corpus is 7.5 characters, and we append two extra characters to the word, we expect the models of order 8-9 to perform the best. This assumption correlates with the results reported by Bisani and Ney [1]. The evaluation consisted of several steps:

The results are shown in the table:

We can observe several trends. First, the error rate drops systematically with the increase of the order of the model. Second, 5 to 10 conversion results seem to give the best variation

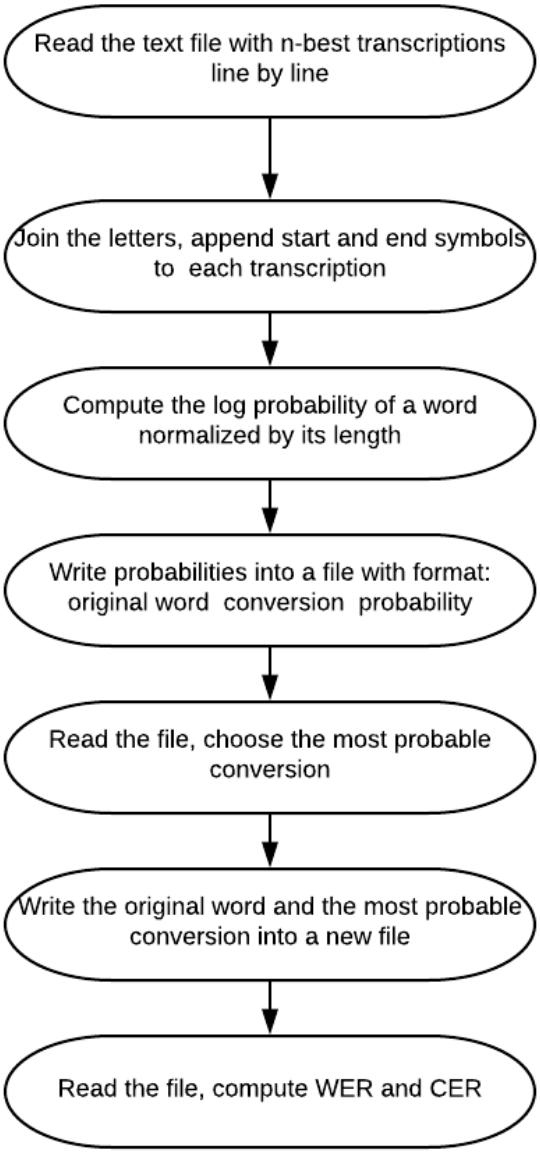

Figure 2.6: The process of probability computation and model evaluation

without increasing confusability of the data. At 20 conversions, the error rates go up. Third, there is an observable improvement in the error rates in comparison to the baseline model: the error rate on 7-grams is about 10-12% lower for 5 and 10 options. Raising the order of the model to select the best sequence does not bring any considerable benefits after 7-gram model. We assume that raising the order of the joint-sequence model during the conversion and applying the Kneser-Ney model to select the best sequence could lower the word error more.

Stupid backoff model does not improve the choice of the best conversion as this model: the 5-gram model applied to 20 -best options gives 67% WER with the discount d=0.4, and 63% with the discount of 0.75. The 8-gram model on the same data achieved 45% error rate against 39 for achieved by the model with Kneser-Ney smoothing.

Another assumption we had is that the accuracy of the selection of the best sequence depends on the order of the model. We applied three different orders to the words of length less than 4, less than 6 and less than 8: the results were exactly the same as for regular model with

P2G conversion:

Precision using the LM: d = 0.75 Kneser-Ney smoothing

	Word error, %				Character error, %			
	5 best	10 best	20 best	50 best	5 best	10 best	20 best	50 best
4 gram	63	70	76	85	8	9	10	11
5 gram	47	51	49	65	6	6	6	8
7 gram	35	36	39	47	4	4	5	6
8 gram	35	36	39	46	4	4	5	6
9 gram	35	36	39	46	4	4	5	6

Figure 2.7: P2G evaluation

Kneser-Ney smoothing. To conclude, the results show that the relatively high error rates for phoneme-to-grapheme conversion can be mitigated by improving the selection of the best result. Kneser-Ney smoothing allowed to decrease the word error rate by about 10% from the original output of the Sequitur converter. The tendency of the accuracy of the conversion to increase with the n-gram closer to the average word length was also confirmed in our experiments.

In the next sections, the experiments with determining word boundary will be discussed.

2.2 Conversion of sequences of words

2.2.1 Two-word input

The majority of speech-to-text applications more commonly deal with sequences of words rather than single words. If the Thus, the converters need to correctly establish the word boundary. As the word boundary does not have any phonemic equivalent, this task becomes extremely challenging. In this project, we experimented with several approaches to deal with the word boundary, working firstly with two word sequences and subsequently with the sequences of n words. Further the respective experiments will be discussed.

2.2.1.1 Training two-words joint-sequence model

In the first experiment, we trained a joint-sequence model on sequences of two words separated by a special word boundary symbol. For this purpose, we randomly concatenated words and corresponding sequences of phonemes in the same data set we used for previous experiments, CMU dictionary. The models were trained up to order 7. We also obtained the lists of 5, 10 and 20 best conversions with the models of order 7. The test results show that this model generally performs twice worse than the single word model:

Model	WER	CER
4 gram	70	11.83%
7 gram	68	10.87%

This can be explained by the fact that this model treats the concatenation of words as a single word. As the average length increases, the order of the model to remember all context grows. Qualitative analysis shows that the place of word boundary in the majority of cases was determined correctly.

2.2.1.2 Selection with Kneser-Ney smoothing

Similar to the experiments with a single word input, we trained a model with Kneser-Ney smoothing based on the two words input and selected the best output from the n-best lists of conversions. The results are displayed below:

Model	WER	CER
5 best	82	10.1%
10 best	84	10.25%
20 best	86	10.85%

The more options in this case provide more opportunities for the confusion of the system.

Training a model with a word boundary seems to be efficient in handling the task of determining the place of the word boundary. However, with a pair of words the risk to convert a word incorrectly increases, and so does the word error.

2.2.1.3 Checking all word boundaries: two-word input

We had an assumption that checking all possible word boundaries and applying the converter to each segment might be beneficial. The algorithm to treat one word is described below:

1. insert a word boundary symbol at every possible place in the phonemic sequence
2. for each pair of words:

   compute the probability of a word according to the language model

   compute the joint probability of two words
3. select the best sequence
4. write the best sequence into a file
5. check if the resulting words are contained in the vocabulary

This model was not sufficiently evaluated due to the limitations of the project.

2.2.1.4 Checking all word boundaries: unknown input

Similar algorithm was applied to process the sequences of an unknown number of words and their phonemic representations. Instead of inserting only one word boundary symbol, we tried all the possible permutations in order to find the most probable ones according to the language model. There are two possible modifications: to split the sequences of phonemes and then apply the conversion with joint-sequence model and vice versa, to convert the whole sequence of phonemes and then try all the possible word boundaries. The second option is notably faster as there. Unfortunately, due to the deficit of time and computational power, we did not manage to process the full test data. Based on several trials on the reduced test data, we may observe the tendency for the model to prefer longer sequences rather than to divide the sequences of letters into a large number of words.

2.2.2 Qualitative analysis

Comparison of the errors pointed out in the evaluation of the model shows the problematic issues for the converter. Non-symmetric correspondences between the letters and phonemes are particularly challenging.

Double vowels and consonants are frequently replaced by a single letter, or vice versa, as in the following example (errors from 20 best list, converted with 7-gram joint sequence model and selected by Kneser-Ney 8-gram model):

original	preferred conversion
ABBAS	ABASE
BEEBE	BIBBEE
BOB	BOBBE
BOBER	BOBBER
FLAB	FLABBE
CLUB	CLUBBE
JEWELS	JEWELL'S
LELLOUCHE	LLELUSH
ADELA	ADELLA

Insertion of silent 'e' or 'h' also seems to be a common case (errors from 10 best list, converted with 7-gram joint sequence model and selected by Kneser-Ney 8-gram model):

original	preferred conversion
GAIN	GAINE
ACROLEIN	ACROWLINE
GIB	GIBBE
MIN	MINE
MONAD	MONADE

A large number of errors are allocated on diphtongs and digraphs: in most cases, these phenomena are captured by the converter, but they often get mismatched:

original	preferred conversion
ALEC	ALLICK
BACKER	BACHER
BLUNK	BLUNCK
BOOKMAN	BUCHMANN
BRUHA	BRUCHA
CANCHOLA	CONCOLA

Some of the errors are caused by irregularities in English spelling: the algorithm prefers a "more English" spelling over the original ones.

In the cases with similar words to the test data being present in training data, the longer n-gram models, that were generally concluded to be more efficient than the lower order ones, replicate the training sequences.

Conclusions

1 Conclusions on the project work

1.1 Discussion of the results

With the joint-sequence model performing well on grapheme-to-phoneme conversion and sufficiently handling out-of-vocabulary words, it was reported to be twice less efficient with phoneme-to-grapheme conversion and detection of a word boundary. The latter two challenges were addressed in this project.

In the experiments, we observed several effects: the effect of the number of conversion options for one word, the effect of the size of the n-gram, the effect of the type of discounting, and the effect of the input size.

The best outcomes were achieved with allowing the joint-sequence model to output 20 best options and apply a character-based n-gram model.

The optimal size of n-gram for both joint-sequence model and character model is close to the average word length in the data (and presumably the language). Varying the length of n-gram depending on the input did not bring significant results: the decrease was no more than 1%.

Two character based language models were compared: a Stupid backoff model and a model with Kneser-Ney smoothing. The latter provided significantly better results dropping the word error rate by around 10%, while Stupid backoff model performed close to the baseline.

The input size plays an important role in the conversion process. The longer the sequence, the more confusion it creates for the system, the word error rate doubled with the concatenation of two words in the input data. We tried several approaches to the detection of the word boundary. Due to the limitations in time and computational resources, we did not inspect this issue in-depth. Nevertheless, the preliminary results show that insertion of a word boundary symbol during the training data is beneficial in terms of determining the place of a word boundary, but the conversion challenges remain the same. Permutations over all possible variant of n sequences for one string of phonemes did not result in observable improvement.

1.2 Completion of the project

To summarize, the objectives established at the beginning of this report were achieved within the span of this project. We explored the performance of the joint-sequence model for phoneme-to-grapheme conversion, manipulated various parameters (size, number of options) and input types and analyzed their performance. We discovered a solution that can potentially improve the performance of the baseline system and proposed several options to handle the inputs of different size. We also compared the results of three evaluation metrics: word error rate, character error rate, and recall.

1.3 Challenges and limitations

One of the main challenges was the training time of the models. The selected Sequitur tool is trained iteratively, so to obtain the model of a sufficient order, we needed to train all the previous ones. As we did not possess the necessary technical equipment to speed up the process, training models consumed a great amount of the project time.

Another challenge was handling the data and managing unexpected encoding problems. Also, as we conducted experiments with multiple models in a short time, it was an uneasy task to organize the data processing to run in parallel in a smooth way.

1.4 Future improvements

There are at least 4 points that should be improves.

First of all, the qualitative evaluation both of the properties of the data and the outputs of different models should be expanded with descriptive statistics.

Second, the experiments conducted on the input of two words and an unknown number of words were not sufficient to conclude on the potential of the proposed techniques. We conducted evaluation only for the model of the highest order performing on two words input. The potential of permutations of all the subsequences of a grapheme or phoneme sequences corresponding to a string of several words was not explored enough.

Third, the character-based model needs to be optimized. In terms of smoothing techniques, the discount coefficients can be tuned with more precision. In terms of implementation, the model could be stored in a more economical prefix tree structure instead of a list of dictionaries.

Forth, we trained and tested the model only on the dictionary data, which are far from the real speech data. Outputs of the automatic speech recognition systems and language models based on large written texts may bring different results.

Finally, the central topic of the project âĂŞ out-of-vocabulary words âĂŞ was addressed intrinsically in the implementation stage with character-based models independent of the reference vocabularies. The rate of out-of-vocabulary words was not controlled in any of the experiments.

2 Personal reflection

The project was highly interesting for the at the beginning as I was hoping to learn more about technical aspects of language processing. It indeed allowed me to improve Python programming skills, which was an unknown field for me prior to the start of the project, and to combat the manifold challenges of creating language models and evaluation tools, preprocessing data, running (and failing) experiments. In the first semester, it was fascinating to learn about automatic speech recognition, another completely new field for me.

The subjects of the first semester, in particular Methods for NLP and Oral corpus, were useful but to my opinion insufficient to lay the groundwork for this project. I was constantly feeling lack of expertise, lack of preparation, and lack of resources during the project. In particular, knowing not only the basic theory of n-gram language modeling but some rationale for the choices made depending on the task, type and size of corpus. I suppose this would have saved me time to actually work on the project rather than trying to fill in the gaps in my background, and I could have achieved more with the task.

Despite the project felt mostly as an endless struggle and I am not completely satisfied with the results, I found it highly beneficial for my development in the field of technology: for the first time, I wrote a program to process large datasets, I managed to implement the solutions proposed by the supervisors, I learnt to be careful with data and to look for optimal solutions. I also encountered the challenges of NLP such as training models for several days on a computer with insufficient memory resource. Besides, it made me realize that out of all fields of NLP speech recognition will unlikely be my priority and I am more interested in NLP that involves linguistic theories and treatment of meaning, which can be regarded as a positive outcome.

There are several things I regret. I regret I did not have enough time to treat meticulously the details both in my code and the process of organizing the process. My involvement in the project was largely affected by personal issues, and I regret I did not manage to experiment with the data obtained directly from the speech recognition systems. I also regret I did not experience team work the way it should be.

Nevertheless, I am satisfied with my participation in this supervised project. I would like to thank my supervisors for being patient and understanding.

Appendices

Appendix A

Functionality implemented in the project:

- 1. Pre-processing of the input data:
  - partition into train and test
  - input file formatting
- 2. Evaluation functions:
  - character error: cer.py, Levenshtein.py
  - recall: recall_char.py
  - word error: reversed_eval.py
- 3. Kneser-Ney model and Stupid Backoff Model:
  backoff_util.py
- 4. Treatment of two word sequences and n-word sequences:
  n_words.py
- 5.Output files with all possible conversions and best choices, according to different models.

Bibliography

[1] Maximilian Bisani and Hermann Ney. "Joint-Sequence Models for Grapheme-to-Phoneme Conversion, Speech Communication". In: (2008). DOI: 10.1016/j.specom.2008.01.00.

[2] *Welcome to Python.org*. May 2019. URL: https://www.python.org.

[3] N. Graves A. Jaitly. "Towards End-to-End Speech Recognition with Recurrent Neural Networks". In: *Proceeding of Machine Learning Research* 32 (1995), pp. 181–184.

[4] Maximilian Bisani and Hermann Ney. "Open vocabulary speech recognition with flat hybrid models". In: Jan. 2005, pp. 725–728.

[5] Carnegie Mellon University. *The CMU Pronouncing Dictionary*. Version 0.7b. URL: http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

[6] Ney H. Kneser R. "Improved backing-off for m-gram language modeling". In: *Proceedings of the IEEEInternational Conference on Acoustics, Speech and Signal Processing* 01 (May 1995), pp. 181–184.

[7] Stanley F. Chen et al. *An Empirical Study of Smoothing Techniques for Language Modeling*. Tech. rep. 1998.

[8] Daniel Jurafsky James H. Martin. "Speech and Language Processing". In: Sept. 2018.

[9] Bill MacCartney. "NLP Lunch Tutorial: Smoothing". In: (Apr. 2005). URL: https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf.