UNSUPERVISED WORD SEGMENTATION FROM NOISY INPUT

Jahn Heymann, Oliver Walter,
Reinhold Haeb-Umbach

University of Paderborn
Department of Communications Engineering
Pohlweg 47-49, 33098 Paderborn, Germany

Bhiksha Raj

Carnegie Mellon University
Language Technologies Institute
5000 Forbes Avenue, Pittsburgh
PA 15213, United States

ABSTRACT

In this paper we present an algorithm for the unsupervised segmentation of a character or phoneme lattice into words. Using a lattice at the input rather than a single string accounts for the uncertainty of the character/phoneme recognizer about the true label sequence. An example application is the discovery of lexical units from the output of an error-prone phoneme recognizer in a zero-resource setting, where neither the lexicon nor the language model is known. Recently a Weighted Finite State Transducer (WFST) based approach has been published which we show to suffer from an issue: language model probabilities of known words are computed incorrectly. Fixing this issue leads to greatly improved precision and recall rates, however at the cost of increased computational complexity. It is therefore practical only for single input strings. To allow for a lattice input and thus for errors in the character/phoneme recognizer, we propose a computationally efficient suboptimal two-stage approach, which is shown to significantly improve the word segmentation performance compared to the earlier WFST approach.

***Index Terms*—** Automatic speech recognition, Unsupervised learning

1. INTRODUCTION

While the task of unsupervised language acquisition from audio recordings is often viewed as direct discovery of repeating patterns that may represent word- or phrase-like structures, *e.g.* [1] [2], arguably a better solution would be one that mediates words through a phonetic inventory. In this situation, the task of unsupervised language acquisition from audio recordings of continuous speech may be divided into two subtasks [3]: a) the discovery of the basic acoustic building blocks of speech, *i.e.* a phonetic inventory, and b) the discovery of the lexical units which manifest themselves as recurring sequences of these basic acoustic building blocks. While the input to the first stage is the raw speech, the input to the second is of categorial nature, the label sequence identifying which of a finite number of acoustic building blocks is present. There are several applications which are characterized by such an unsupervised setup. Besides the mentioned unsupervised learning of an automatic speech recognizer directly from the speech data for low or zero-resource languages [4] [5], there is the semantic analysis of audio data [6] [7], where a high-level transcription of acoustic events can be learned from low-level audio patterns. It may also be helpful as a computational model for early (child) language acquisition.

This paper focuses on task b), the lexical discovery, assuming a known phonetic inventory for the language. A special case of this is the unsupervised segmentation of a written text into words, where, however, the input is a character sequence rather than a sequence of phonemes or other acoustic building block labels. From the perspective of audio recordings, unsegmented text may be viewed as analogous to unsegmented phoneme sequences derived from audio: segmentation would also imply discovering the words themselves. In a practical setting, however, the output of the phonetic discovery unit (task "a" above), will never be error free. Even good phoneme recognizers have a phoneme error rate of well above 10%. Relatively few works exist on unsupervised word segmentation from such noisy (error-prone) input, *e.g.* [8] who try to discover words in phoneme lattices obtained from spoken digits through maximum likelihood estimation, assuming a fixed-vocabulary "multigram" model. However, in practice, the vocabulary size can usually not be assumed *a priori*; on the other hand, additional linguistic structure may validly be assumed. The text analogy for this, then would be the unsupervised segmentation of *noisy* text, with substitutions, insertions and deletions of characters, assuming no prior knowledge of vocabulary, but taking advantage of expected *structure* in the language. This is the problem we tackle in this paper.

We will employ a Bayesian paradigm to assign a probability to a hypothesized segmentation. Since the segmentation must be unsupervised, we can only assume minimal *a priori* information. We will assume that the actual *number* of words to be discovered is not known *a priori*. Consequently, the statistical model we employ must be non-parametric – it must not be specified in terms of a fixed number of parameters, but must rather allow the parameter space to grow with data. We will also assume that words, in a natural setting, follow a power law (Zipf's law) in their occurrrence. The statistical model we employ must capture this power-law occurrence of units. In addition to these, we will also assume that the occurrence of words is predictable, in a statistical sense, and that this predictability is reasonably well captured by an n-gram model.

The above requirements are well embodied in a *nested hierarchical Pitman-Yor* language model (NPYLM). It has previously been demonstrated that by employing this language model to predict probabilities, good segmentation results can be achieved for clean (error free) text [9], as well as for error-free phoneme sequences [9] [10]. Neubig *et al.* [11], in fact, also extend this to *noisy* input, represented by a phoneme lattice, employing Weighted Finite State Transducers (WFSTs) to obtain an elegant and computationally efficient realization. However, as we explain later in the paper, although nominally designed to employ generic n-gram word models, the approach actually breaks down for higher-order n-grams, resulting in degradation of segmentation. In our work, we follow the general approach of Neubig *et al.*, with modifications to rectify the aforementioned problem. This, however, results in considerably increased computational complexity and can, in fact, become intractable if the input consists

of phoneme or character *lattices*. To resolve this problem, we propose a hill-climbing two-stage algorithm, which alternates between extracting the most probable phoneme sequence from the lattice and carrying out word segmentation on that phoneme sequence.
The paper is organized as follows. In the next section we specify our goal and the steps needed to achieve this. In Section 3 we will give a brief introduction on the NPYLM, followed by a description of how the parameters for the model can be estimated in Section 4. This also includes an overview of the WFST-based word segmentation algorithm of [11], its issue, and how it can be solved. In Section 5 we present our two-stage algorithm for word segmentation on noisy input, followed by experimental results in Section 6. Finally, we draw some conclusions in Section 7.

2. UNSUPERVISED DISCOVERY AND SEGMENTATION OF LEXICAL UNITS

Suppose we want to perform a speech recognition for a low or even a zero-resource language. We will assume that a phonetic inventory, which may itself have been derived in an unsupervised manner, is available for the language. In that case, when we perform the phoneme recognition for an utterance (e.g. a recording of the sentence “she had your dark suit”), strong priors (like a dictionary) are likely to be absent. The hypothesized recognition output will be the most probable path through a *lattice* of candidate phoneme hypotheses. Since resources for the language are limited or even not available, the acoustic model will be inaccurate. Hence, the hypothesis will contain substitutions, deletions and insertions of phonemes. For the example, one might end up with ’txeehhedoourdaksoot’. (In this example we have used *characters* as an analogue to phonemes. In general, we will denote the categorical lower-level inputs as “characters”, although it may be either characters, in case of segmentation of text, or phonemes, for segmentation of continuous speech. We will refer to the higher-level structures that are composed from the characters as words.). Not much more can be gleaned from analyzing the individual recordings by themselves, in the absence of other information.

But now assume we have a large number of such utterances. We can now consider the decodes of all the utterances jointly. In such a consideration, one , then, expects to observe consistency between the decodes of these utterances. We can reasonably assume the existence of higher-level units (*e.g.* words), and that these words manifest very similarly in most of their occurrences. If one were to search for such higher-level units, it may be expected that the set of *correct* paths through the lattices for the individual recordings will result in more consistent characterization of these higher-level units than incorrect ones.

Our goal in this work, therefore, is to use this consistency to discover these words and find a hypothesis for the path through the lattices, which will result in fewer errors in the character sequence. In this work, we further assume that the lattice not only contains a better hypothesis, but also that the correct one is embedded somewhere in it.

To utilize the consistency between the words, we need to tackle two tasks:

I. Identify the words by segmenting the character sequence,

II. Find the “correct” spelling for these units in the input lattice. By “correct”, here, we mean a canonical representation that results in the lowest error across all instances of the unit.

For our example, this means that we have to segment it (I.: “txee hed oour dak soot”), and find the correct spelling for each of the identified words in the lattice (II.: “she had your dark suit”).

In order to do so, we need a statistical model, which is able to assign a probability to every possible segmentation for every possible character sequence. We also need a way to effectively estimate the model parameters from the input lattice and to maximize the probability over the segmentation and spelling.

3. NESTED PITMAN-YOR LANGUAGE MODEL

As outlined in the introduction, the model must be able to assign a probability to unknown words based on their spelling and handle an *a priori* unknown number of already discovered words (e.g. be non-parametric). It must also capture the only two constraints we impose:

I. The occurrence of the words follow a power law distribution

II. n-gram structures apply for both, the word, and the character level

The first assumption has shown to be reasonable for most natural, and artificial languages [12]. The second one has been shown to be effective in modeling language data and relies on the fact, that predictability is the fundamental requirement to differentiate anything structured from noise.

As already mentioned, one model which meets these requirements is the NPYLM. It is backed by the Pitman-Yor (PY) process, which is a generalized Dirichlet process governed by two parameters – the discount parameter d and the strength parameter θ – and a base distribution G_0, which is defined over a probability space X. The drawing process for the PY process may be explained through a Chinese-restaurant analogy: at any time the process has a number of “tables”, which can grow infinitely large, and each of these tables has a symbol from X associated with it. This symbol might, for instance, be a word, and X would represent the space of words. In each draw, the new draw (a “customer”) is either “seated” at an existing table and assigned the symbol associated with it, or assigned to a new table. When a new table is selected, the symbol assigned to it is drawn from G_0. The overall process results in draws from a distribution G:

$$G \sim \mathrm{PY}\left(d, \theta, G_0\right). \tag{1}$$

The parameter θ controls how similar this drawn distribution is to the base distribution which itself can been seen as a mean distribution of the drawn ones. Draws from the distribution G obey the power law and hence incorporate the prior knowledge of Zipf’s law.

The n-gram structure is captured by embedding the above in a hierarchical structure. At the n-gram level a separate PY process is instantiated for *every* $n-1$ word context. Each draw employs the PY process that is specific to the current $n-1$ context. The *base* distribution for each PY process at the n-gram level is a $n-2$-context specific PY process. One can view the entire structure as a tree, where the root node gives the unigram probability, its children the bigram probability and so forth. This can be interpreted as smoothing since a certain amount of the probability mass is moved to the shorter context. The degree of smoothing is controlled by the discount parameter d. So instead of one, there are several distributions, one for each context:

$$G\left(\mathbf{u}\right) \sim \mathrm{PY}\left(d, \theta, G\left(\pi\left(\mathbf{u}\right)\right)\right). \tag{2}$$

The notation $\pi\left(\mathbf{u}\right)$ describes the shorter context, e.g. if $\mathbf{u} = (w_{i-1}, \ldots, w_{i-n+1})$ then $\pi\left(\mathbf{u}\right) = (w_{i-1}, \ldots, w_{i-n+2})$. To cope with yet unknown words, another tree is built, this time for the characters, which then serves as the base distribution for the word model.

The model and its underlying sufficient statistics Σ, also called "seating arrangement" in the Chinese-restaurant analogy, are used to calculate the predictive probability of a word w given its context $\mathbf{u}$ and the parameter vector $\Phi = (\mathbf{d}, \boldsymbol{\theta})$:

$$\begin{aligned} \Pr(w|\mathbf{u}, \Sigma, \Phi) &= \frac{c_{\mathbf{u}w\cdot} - d_{|\mathbf{u}|} t_{\mathbf{u}w}}{\theta_{|\mathbf{u}|} + c_{\mathbf{u}\cdot\cdot}} \\ &+ \frac{\theta_{|\mathbf{u}|} + d_{|\mathbf{u}|} t_{\mathbf{u}\cdot}}{\theta_{|\mathbf{u}|} + c_{\mathbf{u}\cdot\cdot}} \Pr(w|\pi(\mathbf{u}), \Sigma, \Phi) \\ &=: \Pr_{\text{local}}(w|\mathbf{u}) + \Pr(\text{FB}|\mathbf{u}) \Pr(w|\pi(\mathbf{u})) \end{aligned} \tag{3}$$

with the obvious definition for $\Pr_{\text{local}}(w|\mathbf{u})$ and $\Pr(\text{FB}|\mathbf{u})$[1].
Again, a Chinese-restaurant analogy may be used to interpret eq. (3). $c_{\mathbf{u}w\cdot}$ describes the counts of word w in the context $\mathbf{u}$. $t_{\mathbf{u}w}$ describes how many "tables" are occupied by this word and states the number of counts which has been added to the shorter context. It is therefore an indicator of how much smoothing is applied. $\mathbf{d}$ and $\boldsymbol{\theta}$ are the vectors of discount and strength parameters of the Pitman-Yor processes at the different levels of the hierarchy and each parameter is shared along the same context length. The $\cdot$ indicates a marginalization, so $c_{\mathbf{u}\cdot\cdot} = \sum_w c_{\mathbf{u}w\cdot}$ is the count of all words in the context $\mathbf{u}$. In the generative perspective $\Pr(\text{FB}|\mathbf{u})$ is the probability for assigning a new table for a new draw. The parameters Φ are sampled after each iteration as is explained in [13].

With the help of eq. (3) we are able to assign a probability to a hypothesized segmentation and spelling $\tilde{\mathbf{w}}^{(i)} = \left[\tilde{w}_1^{(i)}, \ldots, \tilde{w}_k^{(i)}\right]$ of the sentence $s^{(i)}$:

$$\Pr\left(\tilde{\mathbf{w}}^{(i)}\right) = \prod_{j=1}^{k} \Pr\left(\tilde{w}_j^{(i)}|\mathbf{u}_j\right) \tag{4}$$

In order to give the first words a context, we prepend $n-1$ start sentence symbols.

4. ESTIMATING THE MODEL PARAMETERS

Now, that we have found a model which fits our needs, we need a way to estimate its parameters from our given input lattice and to maximize the probability of our segmentation and character string.

4.1. With text input

We first consider the case of a noise-free text input, where the spelling of every sentence is correct. This is the standard task for word segmentation. Our goal is to find the most probable seating arrangement (e.g. segmentation) for the given input: the words, and the counts and occupied tables for each of them in each restaurant (PY process). One can imagine that the number of possible seating arrangements is too large to calculate the probability for all of them. To overcome this issue, Mochihashi *et al.* [9] propose to use a technique called blocked Gibbs sampling to get samples from the distribution. Each sentence is considered as one block. Now, in order to get a new segmentation for a sentence $s^{(i)}$, first its statistics is removed from the language model by decreasing the counts the sentence has contributed to. Then, the language model is used to calculate the predictive probability for every possible segmentation $\mathbf{w}^{(i)}$ of that sentence. Finally, a sample segmentation for this sentence is drawn and the resulting words are added to the language model, updating the seating arrangement. To get a sample, forward filtering/backward sampling is used, which is a dynamic programming approach to allow for efficient sampling. Employing these techniques, it is possible to estimate the language model parameters along with the segmentation.

[1]Note that, in the following, we will leave out the dependencies on the sufficient statistics and the parameters in the notation

Table 1. Comparing the segmentation results for WSJCAM0. (1) Our results with a bigram word model, (2) Neubig et al. with a bigram word model, (3) Our results with a unigram word model (4) Neubig et al. with a unigram word model

	(1)	(2)	(3)	(4)
Precision	71, 1%	29, 6%	64.5%	54, 3%
Recall	60, 5%	46, 6%	51.4%	50, 2%

4.2. With a lattice input

In case of a lattice input, the number of possible seating arrangements is even bigger. Instead of every possible segmentation, we now also have to consider every possible segmentation for every path through the lattice. To do so, [11] proposes a method based on WFSTs similar to the ones used in ASR to carry out the task. Again, blocked Gibbs sampling is used to get samples from the distribution. But here, after removing the statistics from the language model, not only every possible segmentation, but also every possible spelling for a sentence is considered and assigned a probability. Then, a sample is drawn from this, jointly sampling the segmentation and the character sequence of the sentence. The method is elegant since it requires only a minimal amount of changes to a commonly used lexicon and language model and can therefore be directly integrated in a speech recognizer training workflow. Additionally it relies on efficient graph algorithms. However, implementation itself results in an implicit restriction: it cannot accurately assign predictive probabilities according to the generative model (3) for higher-order (> 1) n-grams, resulting in degradation of performance for larger n.

Empirical evidence for this is presented in Tab. 1. The table compares results obtained with the implementation of [11] (using their own code from [14]) with an update we propose (described later in this paper) on text prompts of WSJCAM0, with white spaces and punctuation removed, under identical parameter settings (100 iterations, 8-gram character model). The first and third column show results of our implementation, while the second and the forth column are obtained with the software of [14], using a bigram (1st and 2nd column) and unigram (3rd and 4th column), respectively. As can be seen, the bigram results of [14] are worse than the results obtained with our implementation; and they are even worse than the unigram results obtained with [14]. Also, the results obtained with our update are comparable to those we previously achieved [10] using an approach which does not rely on WFSTs and has been proposed in [9].

To understand the reason for this drop in performance, let us consider the model of [11] more closely. The algorithm is implemented using a WFST that is turn composed of two WFSTs: One is the lexicon, which is responsible for storing already discovered words and also for generating yet unseen ones. The transducer can either transduce a sequence of characters into a known word, or, for novel words, it can pass the characters through as they are and in-

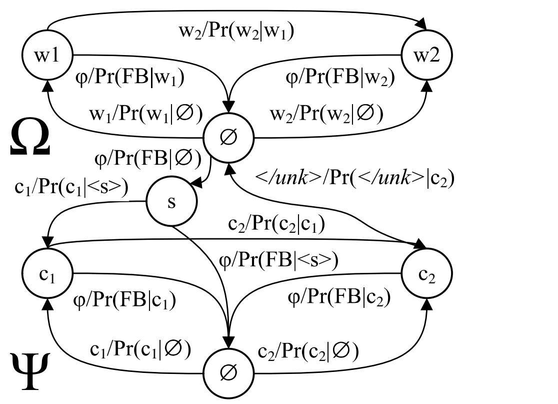

Fig. 1. Grammar consisting of the word model Ω and the spelling model Ψ for a bigram case

sert a word-end tag after at least one character has passed. The other transducer is the language model, or *grammar* G. Its task is to assign a probability to a segmentation according to the model. The transducer may either assign probabilities to already known words (*word model* Ω), or employ the character transducer to assign probabilities to novel character sequences (*spelling model* Ψ).

Fig. 1 illustrates this for a bigram model. Each node represents a chinese restaurant. The edges are weighted with the predictive probability for each input which can be calculated using equation (3). When no outgoing edge for a given input is available, the path to the shorter context is taken till a state with an appropriate edge is reached. Each of these transitions has the fallback probability $\Pr(\text{FB}|\mathbf{u})$ assigned to it. To leave the character model, a *</unk>* tag is required. This tag is used to mark the end of a new hypothesized word and, seen from a generative viewpoint, enables the character model to not only produce characters but also words.

Now consider first the case with a unigram language model and an m-gram spelling model. Suppose we want to calculate the probability of w_1, which consists of the character sequence $\mathbf{c}_1$, and we have seen this word before. From eq. (3) we have

$$\begin{aligned}\Pr(w_1|\emptyset) = &\frac{c_{\emptyset w_1 \cdot} - d_{|\emptyset|} t_{\emptyset w_1}}{\theta_{|\emptyset|} + c_{\emptyset \cdot\cdot}} \\ &+ \frac{\theta_{|\emptyset|} + d_{|\emptyset|} t_{\emptyset \cdot}}{\theta_{|\emptyset|} + c_{\emptyset \cdot\cdot}} \Pr(\mathbf{c}_1) \end{aligned} \tag{5}$$

Using the WFSTs, we would get both w_1 and $\mathbf{c}_1$ from the lexicon transducer as the input for the grammar transducer. Hence we would have two paths and get the probability

$$\hat{\Pr}(w_1|\emptyset) = \Pr(w_1|\emptyset) + \Pr(\text{FB}|\emptyset)\Pr(\mathbf{c}_1), \tag{6}$$

where the first contribution is from the word and the second from the character model. Comparing with eq. (5), the character model contribution is wrong: since word w_1 is known, it need not also be assigned a probability by the spelling model.

To overcome this issue, Neubig *et al.* set the base probability $\Pr(\mathbf{c}_i)$ for each word to zero when calculating the probabilities for the edges of the word model. Doing so, the probability provided by the word transducer reduces to $\Pr_{\text{local}}(w_1|\emptyset)$ and eq. (6) delivers now the correct probability, eq. (5).

This, however, only holds true for a unigram word model where both paths end in the same state - the root state. Therefore the base probability always gets added in the following sampling step. But for any n-gram model of greater order, the path with a word symbol will end in a word context node, while the path with the character sequence will still end in the root node. Since the states are different, the base probability will not get added before the multiplication with the probability of the next word, leading to false predictive probabilities. As a consequence, already discovered words are assigned a higher probability than the one the model proposes. As we see in the results, this leads to an oversegmentation of the text, resulting in a smaller vocabulary, containing often-used sub-word units.

5. UNSUPERVISED WORD SEGMENTATION FROM LATTICE INPUT

For our work, we follow the basic paradigm of [11], however we propose an alternate design for the lexicon to avoid the issues mentioned in Section 4.2. Unfortunately this is not a trivial task. Our goal is to get a lattice which represents all possible segmentation with all possible spellings of an input sentence using both, known and unknown words, without hypothesizing an already known word again. This task cannot be carried out by a generic transducer since we need to know the string of a potential word in advance and cannot decide whether it is going to be a new or a known word just by the character at the beginning. Therefore our algorithm analyses each possible string and uses a hash table to decide if it is a known or a new word. The lexicon transducer is then built such that it transduces an unknown character string and appends the end word symbol only if it is not an already known character string.

Additionally, we now include the base probability for every word when calculating the probabilities for the edges of the word model. This is due to the fact, that we don't represent a known word with its character sequence in the input of the grammar anymore. Therefore, we don't get the additional term in eq. 6 and can use the eq. 3 without any modification.

However, this is computationally more costly compared to the implementation by Neubig *et al.* since the lattice with every possible segmentation is much wider and has significantly more states. Therefore the composition with the grammar is much more time consuming. Nevertheless it is necessary in order calculate the probabilities according to the underlying model and we will use this approach.

To estimate the model parameter along with the segmentation and spelling, we have to consider every possible segmentation and spelling. Even when using blocked Gibbs sampling, we still need to consider every possible segmentation and every possible spelling for a sentence. Since the possibilities grow exponentially with every alternative character, this becomes computationally overwhelming with the presented correction. We therefore choose a different approach, which does not sample the segmentation and the character sequence jointly. Instead, we alternate the following steps each iteration:

I. Given a single character sequence at the input, conduct unsupervised word segmentation and learn the word and spelling model alongside segmentation

II. Given the spelling model of step I, compute the most probable character sequence from the input character lattice.

5.1. Step I: Estimating the segmentation and the language model

The input for this step is the current most likely character sequence $\hat{\mathbf{c}}^{(i)}$ for every sentence $s^{(i)}$ where $i = 1, \ldots, N$ indicates the sentence number. We now want to draw a sample segmentation for each

Best shot input:
POWEYFSNKNZIMLCIAFBNCNUINLSVIVAHEVGGVCE OQRPHBOISSRXTHHIZBUQSIENASELDBYPMUMRTOR YIRTTIONRTJAGNFAATDNTRZTPBATEKTLDINGOMI AJV
Proposed after 25 iterations:
POWER FINANCIAL IS A FINANCIALSERVICES CONCERN THAT IS SIXTY NINE PERCENT HELD BY POWER CORPORATION OF CANADA A MONTREAL BASED HOLDING COMPANY
[14] after 25 iterations:
POWER FINANCIAL CS C FI NANCIAL SERVIC ES CONC ERN THAT IS SIX TY NINE PERCENT HEL DBY POWER CORPOR ATION OF C ANAD A A MONT REAL BA SED HOL DING COMPANY

Fig. 2. The best shot input at the beginning and the output after 25 iterations for the proposed method and the implementation [14]

sentence and, by doing so, update our estimation of the language model parameters. Since the input is a single character string, we only have to consider all possible segmentation. We can use the same techniques as described in [9] or, for a WFST implementation, in [11] to get a sample segmentation and update our language model.

5.2. Step II: Maximizing the probability of the character sequence

In this step we aim to find the best character sequence $\hat{\mathbf{c}}^{(i)}$ for each sentence given our current knowledge of the spelling model.

Therefore we consider all possible segmentation for every character sequence for the sentence $s^{(i)}$. Since we will only use the spelling model here, we don't need to mark any string as a known word. This makes the process much faster while still giving a good estimate as we will see later on in the results section.

In order to compute the predictive probabilities, we remove the statistics from sentence $s^{(i)}$ from the spelling model. After that, we use it to *approximate* the probabilities for every possible segmentation for every possible character sequence, choosing the most likely sequence as the input for the next iteration.

Note that by using the spelling model we use our knowledge about the current segmentation and therefore implicitly incorporate higher level units, e.g. words. This would not be the case if we used statistical models just based on the possible character sequences.

6. RESULTS

We evaluated our model and compared its performance with the implementation from Neubig *et al.* [11].

The experiments were conducted with lattice generated from text input. We used the text prompts of the WSJCAM0 [15] acoustic model training data, which is a subset of 5612 unique sentences, containing a total of 95453 words (10660 unique ones), of the Wall Street Journal (WSJ0)[16] acoustic model training database. First, all word delimiters and punctuation were removed from the text prompts and then all characters were transformed to upper case. This resulted in the error-free character input. To generate a lattice, the procedure described in fig. 4 was used. An extract of such a lattice is shown in Fig. 5. Note, that this way, among the many character string alternatives the lattice describes there is also the true, i.e, error-free character string.

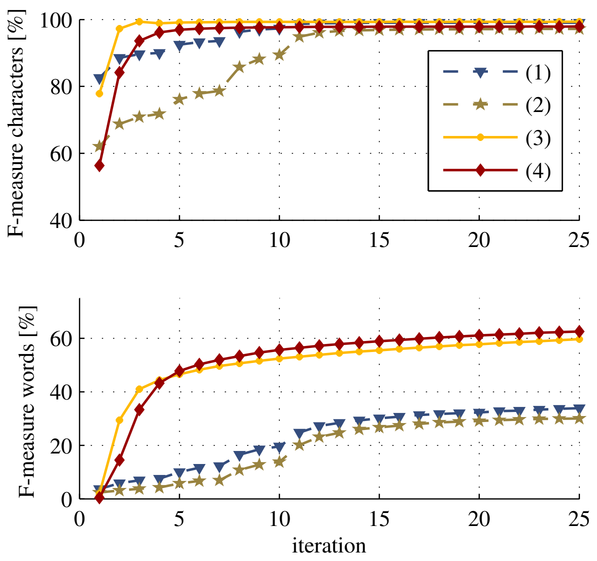

Fig. 3. F-measure for the characters and words over iterations for different strength of noise: *(1)* [11] with $X = 50$, *(2)* [11] with $X = 100$, *(3)* proposed with $X = 50$, *(4)* proposed with $X = 100$

```
1: procedure GenerateInputLattices
2:     for each c in $X\%$ of the correct characters do
3:         $p \sim$ unif $([0 \ldots 1])$
4:         $\tilde{c} \sim$ unif $([c_1, \ldots, c_n, ins, \epsilon])$
5:         if $\tilde{c} == ins$ then
6:             $\hat{c} \sim$ unif $([c_1, \ldots, c_n])$
7:             add $c$ with weight $p$
8:             add $[c\hat{c}]$ with weight $1 - p$
9:         else
10:            add $c$ with weight $p$
11:            add $\tilde{c}$ with weight $1 - p$
12:        end if
13:    end for
14: end procedure
```

Fig. 4. Procedure to generate the input lattices

The two-stage iterative word segmentation algorithm was now applied to the input lattice. All experiments were carried out with a bigram word model and an 8-gram character (spelling) model. We let the algorithm run for 25 iterations, which are enough for the algorithm to converge. The improvements afterwards were barely noticeable. As for the error measure, we use the F-measure on both, the word, and the character level. The first one reflects the quality of the segmentation *and* the spelling, counting only correctly spelled words as true positives, while the later one gives an impression of the overall spelling, counting all correct characters as true positives, while ignoring the segmentation. To take deletions and insertions into account, we used the minimum edit distance for alignment before calculating the F-measure.

Fig. 2 depicts an example. The first string is the most probable character string ('best shot') in the input lattice at the beginning of the iterations, while the second and the third depict the output after 25 iterations gained with our implementation and [14] respectively. While both methods are able to recover the true character sequence

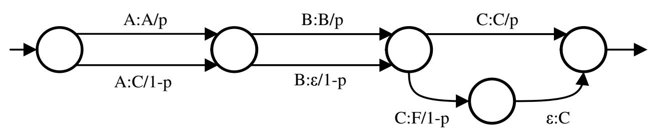

Fig. 5. Example for the input lattice with substitution, deletion and insertion

nearly perfectly (the flaw does not affect the spelling model after all), there is a noticeable difference regarding the words. While our implementation is able to recover most of them correctly, the one by Neubig *et al* oversegments it into smaller sub-word units.

Fig. 3 presents the quantitative evaluation results for the proposed algorithm and the algorithm from [11] for two different degrees of the severeness of the noise. For $X = 100$, for every correct character, there is an alternative randomly chosen incorrect entry, while for $X = 50$ this holds for every other character. These results confirm the previous statements. Both methods are able to retrieve the correct character sequence nearly perfectly. But our algorithm outperforms the other one in terms of word discovery / segmentation, reaching an F-measure of $64,6\%$ versus 30.69% for $X = 100$. The ability to recover the character sequence confirms the results reported by Neubig *et al*, that the phoneme error rate can be significantly improved [11].
We also conducted some experiments with a unigram word model to evaluate our approximation. Surprisingly, our algorithm achieves a word F-measure of $50,1\%$ ($98,8\%$ on the character level), while the program from Neubig *et al.* [14] achieves $41,0\%$ ($96,0\%$). Both experiments were conducted with $X = 100$ and an 8-gram spelling model. While the exact reasons for this remain unclear for now, it is a good indication, that the losses utilizing an alternating optimization versus a combined optimization are not very big. These results also confirm that the segmentation can be significantly improved by using a higher order word model, justifying our efforts to find a solution to apply a higher order model in the case of uncertainties in the input data.

7. CONCLUSIONS AND OUTLOOK

In this paper we have presented an unsupervised word segmentation algorithm that is able to cope with errors in the input label sequence. To this end, the input sequence is expanded to a lattice to allow for alternative character sequences, among which the true, error-free sequence is assumed to be. Then an iterative WFST-based word segmentation algorithm is proposed where in each iteration one alternates between estimating the most probable character sequence for a given language model and estimating the word segmentation, and the language model given the most probable character sequence. Significantly improved word segmentation performance was observed on a noisy text input compared to a method from the literature. In future we will apply the algorithm to the lattice output of a phoneme recognizer to enable lexicon discovery and language model estimation from continuous speech. We further plan to investigate if the overall system can be improved if the results of the word segmentation are fed back to the phoneme discovery stage.

8. ACKNOWLEDGMENTS

This work was in part done at and in cooperation with the Language Technologies Institute of Carnegie Mellon University, supported by Deutsche Forschungsgemeinschaft under contract no. Ha 3455/9-1 within the Priority Program SPP1527 "Autonomous Learning" and partially funded by NSF grant 1017256

9. REFERENCES

[1] A. Park and J. Glass, "Unsupervised pattern discovery in speech," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 1, 2008.

[2] L. ten Bosch and B. Cranen, "A computational model for unsupervised word discovery.," in *INTERSPEECH*, 2007.

[3] A. Jansen, E. Dupoux, S. Goldwater, M. Johnson, S. Khudanpur, K. Church, N. Feldman, H. Hermansky, F. Metze, R. Rose, et al., "A summary of the 2012 jhu clsp workshop on zero resource speech technologies and models of early language acquisition," in *Proceedings of ICASSP*, 2013, vol. 2013.

[4] James Glass, "Towards unsupervised speech processing," in *Information Science, Signal Processing and their Applications (ISSPA), 2012 11th International Conference on*. IEEE, 2012.

[5] J. Schmalenstroeer, M. Bartek, and R. Haeb-Umbach, "Unsupervised learning of acoustic events using dynamic time warping and hierarchical k-means++ clustering," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[6] Sourish S. Chaudhuri and B. Raj, "Unsupervised structure discovery for semantic analysis of audio," in *Advances in Neural Information Processing Systems 25*, 2012.

[7] D Harwath, T Hazen, and James R Glass, "Zero resource spoken audio corpus analysis," in *Proc. of the IEEE Inter. Conf. on Acoust., Speech and Signal Proc.(ICASSP)*, 2013.

[8] J. Driesen and H. Van Hamme, "Improving the multigram algorithm by using lattices as input," 2008.

[9] D. Mochihashi, T. Yamada, and N. Ueda, "Bayesian unsupervised word segmentation with nested pitman-yor language modeling," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, 2009.

[10] O. Walter, R. Haeb-Umbach, S. Chaudhuri, and B. Raj, "Unsupervised word discovery from phonetic input using nested pitman-yor language modeling," ICRA Workshop on Autonomous Learning, 2013.

[11] G. Neubig, M. Mimura, and T. Kawaharak, "Bayesian learning of a language model from continuous speech," *IEICE TRANSACTIONS on Information and Systems*, vol. 95, no. 2, 2012.

[12] C. Manning, Christopher D, and H. Schütze, *Foundations of statistical natural language processing*, MIT press, 1999.

[13] Yee Whye Teh, "A bayesian interpretation of interpolated kneser-ney," 2006.

[14] G. Neubig, "latticelm," Apr. 2013.

[15] J. Fransen, D. Pye, T. Robinson, P. Woodland, and S. Younge, *WSJCAMO corpus and recording description*, Citeseer, 1994.

[16] D. Paul and J. Baker, "The design for the wall street journal-based csr corpus," in *Proceedings of the workshop on Speech and Natural Language*, 1992.