Embedding grammar to N-gram Language Model Based on Weighted Finite-State Transducer

Author 1
XYZ Company
111 Anywhere Street
Mytown, NY 10000, USA
author1@xyz.org

Author 2
ABC University
900 Main Street
Ourcity, PQ, Canada A1A 1T2
author2@abc.ca

Abstract

Building a language model usually requires massive training corpus, which labor-consuming and time-costing. And it is impractical to construct a language model that covers all the spoken language, especially for some special domains. In this paper, we combine the class-based and word-based language model based on weighted finite-state transducer using our grammar embedding method, and the combination is controlled by a merge weight. The class we used in our work is a type of named entity, such as address name, person name etc. We test our method on a domain-specific test set and our approach can dramatically improve the recognition rate of the low-frequency or unseen words. In addition, we explore the relation between the optimal merge weight and the experiment settings.

1 Introduction

The n-gram language model (LM) building is one of the key techniques in Automatic Speech Recognition (ASR) for providing a statistical analysis in decoding results prediction. Traditionally, a LM is trained on large corpora by counting the appearance of the n-grams and estimating their probability distributions. It is then converted into a state transition graph using Weighted Finite-State Transducer (WFST) technique and finally composed with other components of the speech recogniser. However, sometimes it is impractical to construct a LM that covers all the spoken language, especially for some special domains. For example, given a large vocabulary such as a long list of named entity (NE), for the names that never occurs in the training corpus, it is almost impossible to recognise them from the word-based LM. Moreover, when a complex grammar pattern like phone number series is required, it is very hard to find a corpus that contains every possible phone number. In short, a lack of scalability is the limitation of the traditional word-based LMs in ASR, and it should not be solved by only adding more training data.

Therefore, in order to train a extensible LM, one solution is to use the class-based n-gram model. It is especially useful when the model is expected to contain a variety of entity names from the same type of class, such as person names or address names. More importantly, the scalability of class-based LM allows us adding more new entity names into the class cluster or changing grammar rules without rebuilding the main LM. In our work, we denote each class as a tag, such as '<address>' for the class of address names. Each class is described in some specific grammar patterns, such XML and JSpeech Grammar Format (JSGF)[1]. And we can use the tags to represent not only some word sequences but also grammar patterns with more complicated structures. We can combine the class-based LM and the grammar patterns together based on weighted finite-state transducer (WFST) using our grammar embedding method. The composite WFST graph is called a tag LM. By doing this, this tag LM will take the role of a grammar component in the speech recognition system, such as the grammar component G in KALDI's HCLG framework (Mohri et al., 2008; Povey et al., 2011).

In this paper we first present the method for building the WFST graphs from the training data and the JSGF pattern, then propose a embedding technique for merging the grammar patterns into the main LM graph guiding by a merge weight. We explore the relation between the merging weight and the experiment settings and discuss how it effects the recognition process on words or word sequences specified by the grammar patterns by evaluating on a domain-specific test set. The result shows that our approach can dramatically improve the recognition rate of the low-frequency or unseen words(sequences). We also

[1] http://www.w3.org/TR/jsgf/

developed an open source toolkit for the technical implementation[2].

The remainder of this paper is structured as follows. Section 2 discusses relevant works from class-based LM training to FST implementation in ASR. Section 3 presents our grammar embedding method. Section 4 empirically evaluates the performance of our work and Section 5 concludes.

2 Related Work

The performance of the class-based LM has been studied within literature of language model training techniques and applications. Typically, the class-based LM has a better outcome than the word-based LM in the cases of large vocabulary (Samuelsson and Reichl, 1999), lacking in training data (Ward and Issar, 1996) or word sequence patterns involved (Georges et al., 2013). By using the class-based LM, there are many researches show decrease in perplexity (Ward and Issar, 1996; Samuelsson and Reichl, 1999) and word-error rates (WER) (Georges et al., 2013).

(Schalkwyk et al., 2003) and (Georges et al., 2013) introduced two techniques for decoding the embedded graph dynamically during runtime. They are both done on-the-fly. (Schalkwyk et al., 2003) preposed a method for embedding the dynamic grammars into its CPLG framework. which is a WFST model that has been composed from the language model (G), lexicon (L), phone (P), and context-dependency (C). When a tag of class is encountered during decoding process, the corresponding dynamic grammar is called recursively. (Georges et al., 2013) took a similar way to embedd the grammar graphs, where the grammar graphs are searched when the tag is reached. However, it is distinguished from (Schalkwyk et al., 2003) by embedding the grammar graphs to the language model G. The reason why both (Schalkwyk et al., 2003) and (Georges et al., 2013) prefer on-the-fly method to full expansion is that fully expanding the grammars may cause the resultant graph too large. However, without full expansion, the optimisation technique (e.g determinisation, minimisation, etc.) cannot be applied to the graph unless implementing extra structures. Therefore the recursive transition networks and the nesting technique are implemented respectively.

In short, using class-based LM can improve the performance on speech recognition. However, it requires full expansion on the decoding framework in order to the determinisation. which is very costly. Otherwise, one have to exploit additional techniques to perform the decoding process.

In our research, we propose a technique that links the grammars to the language model G before G is composed with other components(H, C and L), with additional disambiguated nodes and approximation method, the graph can be determinised and available for further optimisation. More details will be discussed in Section 3.

3 Methodology

Each class if defined by a rule in a grammar file. Later we use the low-frequency items in all the classes to tag the training corpus, that is substituting the word strings in trainning corpus with a class token. A n-gram language model is generated for class tokens using the tagged training corpus. Compiling the n-gram language model and the grammar to two WFSTs and merging the WFSTs to a integrated WFST parameterized by the merge weight. After the merging procedure, the integrated WFST serves as a component(G) of constructing HCLG. The procedure of our approach is as follow:

1. build a grammar that defines all the classes.
2. tag the training corpus and train a class based n-gram language model.
3. merge class based n-gram language model and the grammar to a intergrated WFST.

3.1 Building Grammar

We define the classes as different types of NE, that is each type of NE is a class. The items in each class are a cluster of NEs belonging to the same type and each cluster of items is predefined in a corresponding NE list. Then each class in specified in a grammar file as one rule. The format of the grammar we used is JSGF:

```
grammar NE_grammar;

public <LOCATION> = LOC1 | LOC2 | LOC3;

public <PERSON> = PER1 | PER2 | PER3;

public <ORGANIZATION> = ORG1 | ORG2 |
  ORG3;
```

In the above example grammar, three classes (LOCATION, PERSON and ORGANIZATION) are defined. The LOCATION class contains three items (LOC1, LOC2 and LOC3), each item can be a single word or word sequences.

3.2 Tagging and Training Class-based LM

We used our parser to tag the training corpus referring to the NE lists. The parser replace the word strings with the class token. We then generate an ARPA n-gram LM using SRLM toolkit (Stolcke and others, 2002). There is a special implementation

[2] http://cslt.riit.tsinghua.edu.cn/mediawiki/index.php

consideration. We should filter the NE lists since there are some high-frequency NE in the lists. High-frequency words can be trained sufficiently using word-based n-gram LM and there is no need to incorporate them in the class cluster. If high-frequency words are reserved and there is no proper probability assignment in the grammar, the recognition rate of high-frequency will deteriorate. So our parser will count the frequencies of each NE and abandon the high-frequency NEs, and the grammar should be altered correspondingly. As a result, we use the filtered low-frequency NEs to tag the corpus and to build the grammar.

3.3 Grammar Embedding Method

We first convert the JSGF file to FSG using sphinx toolit[3]. Then we use OpenFST toolkit[4] to compile the FSG to a grammar WFST. Figure 1 illustrates a simple grammar WFST. A class-based LM WFST is generated by convert the ARPA LM created in section 3.2 to WFST using KALDI (Povey et al., 2011). Figure 2 exemplifies a class-based LM WFST. The integrated LM WFST is presented in Figure 3.

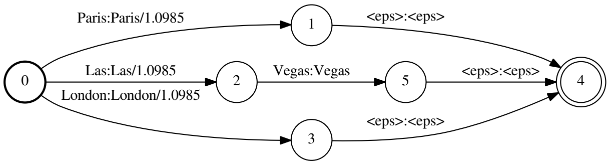

Figure 1: Grammar WFST

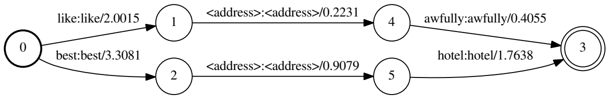

Figure 2: Class-based LM WFST

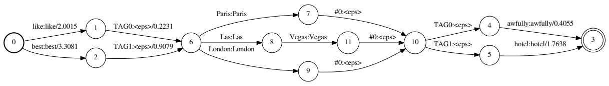

Figure 3: Integrated LM WFST

In the grammar WFST, the input or output word (sequence) of each path from the entrance sate 0 to the exit state 4 corresponds to an item of a class (e.g. <address>) in JSGF. And the weights on all the transitions leaving the entrance state 0 equal. Strictly, these weights should be distributed according to the the n-grams probability of each item. Since we tag the training corpus with low-frequency items, their probabilities vary slightly. So equal weight is approximate but reasonable.

In the class-based LM WFST, there are two word seqences each contains a tag <address>. We embed Figure 1 into Figure 2 and Figure 3 is obtained. We do not replace each tag in Figure 2 with the grammar WFST. But all the tag in Figure 2 share one grammar WFST. We choose a grammar graph sharing method and give up the full expansion method because full expansion will give rise to a very large resultant WFST, which is very difficulte to handle. The specific precedure is presented in Algorithm 1.

Algorithm 1 Grammar Embeding

```
set $i = 0$
for each transition $t = (p[t], < address >, < address >, w[t], n[t])$ in Figure 2
repeat
    connect $p[t]$ with $entr[g]$ and generate a new transition $tt = (p[t], TAGi, < eps >, w[t] * factor, entr[g])$
    connect $exit[g]$ with $n[t]$ and generate a new transition $ttt = (exit[g], TAGi, < eps >, 0, n[t])$
    set $i = i + 1$
until Done
```

In Algorithm 1, $entr[g]$ denotes the entrance state in Figure 1 while $exit[g]$ denotes the exit state in Figure 1. Additionally, $factor$ denotes the merge weight, which controls how easily the decoder steps into the grammar WFST. The smaller merge weight is, the more likely decoder steps into the grammar WFST, hence more likely to decode the words (sequences) in JSGF. In the previous illustration, the merge weight is 1. So the weight from state 1 to 6 in Figure 3 is same with the weight from state 1 to 4. They are both 0.2231.

The input label 'TAGi' is added on the generated transitions to guarantee the integrated LM WFST is determinizable. It utilizes the twin property (Mohri et al., 2008) of WFST.

4 Experiment

In our experiment, we use one type of NE (address name) and one corresponding address list. The items in the address list are treated as one class while other words are defined as individual classes as themselves. We execute the work flow described in section 3 and get a HCLG WFST and then evaluate the it on a test set created by ourselves.

4.1 Experiment Setup

In order to evaluate the perfomance of our method, we construct a test set. It contains 12 person's record of total 120 sentences. Each sentence contains a

[3] http://www.speech.cs.cmu.edu/sphinx

[4] http://www.openfst.org

address name. The preceding 30 sentences have high-frequency address name in each sentence, and the middle 40 sentence have low-frequency address name which is listed in the JSGF grammar file in each sentence, while the posterior 50 sentence have address name that is unseen in corpus but added to the grammar. We add 10 unseen address names to the grammar, and each unseen address name have 5 test sentences. We want to see whether our method can help us recognize the low-frequency address names and the unseen address names on condition that don't influence the recognition of other words.

The training corpus we used has a total size of 64MB and consists of about 1.4 million sentences and 11.5 million words. Then we use it to train a 5-gram language model. The speech database we used contains 1400 hours of speech data. The 40-dimensional Fbank feature was used in DNN training and recognition. The lexicon we used has 150K words.

4.2 Baseline

We set up a baseline experiment to compare with our method. We use the training corpus to train a word based 5-gram language model. Using this language model to construct HCLG and testing on the test set. The result is reported in 1.

WER	30_ER	40_ER	50_ER
20.66	6	16	32

Table 1: Result of baseline. 30_ER, 40_ER and 50_ER means the error number of 30 high-frequency address names sentences, 40 low-frequency address names sentences and 50 unseen address name sentences

In the baseline experiment, the word error-rate is 20.66%. There are 6 errors in recognizing the high-frequency address names, 16 errors in recognizing the low-frequency address names and 32 errors in recognizing the unseen address names. Note that there are 18 correct case when recognizing the unseen address name. The reason for this result is that the grammar contains 2 long address names. Although the long address names don't appear in training corpus, but each single word composing the long address names does. The decoder will recognize the long address names by backing off to unigram.

4.3 Tag LM

We conduct 3 experiments to show our method works and to find the relationship between the optimal merge weight and the number of address names in the grammar.

In experiment 1, we extract 490 low-frequency address names that appear in the training corpus. We use these 490 address names to tag the corpus, and there are total 1369 sentences that matches the address names. Then we combine the 490 low-frequency and 10 unseen address names and the grammar is determined. Using our method to build several class based HCLGs with different merge weight. Table 2 presents the result. From the result we can see that our method can largely improve the recognition rate of low-frequency and unseen address names, especially when the merge weight is small. But when the merge weight is smaller than a threshold, the WER is high. The explanation for this phenomenon is that when the merge weight is small, it's more likely for the decoder to recognize ordinary words as address names in the grammar. When the merge weight is high, the decoder is less likely to recognize the address names. One extreme is that when the merge weight is very big, the result of our method is same with the baseline. There is a trade-off between the recogniztion rate of the low-frequency or unseen address names and other ordinary words. In experiment 1, the optimal merge weight is 0.6. The criterion to judge whether the merge weight is good is that 40_ER and 50_ER is small on condition that 30_ER is not higher than baseline. Low 30_ER gurantees the recognition rate of ordinary words is not obviously influenced.

merge weight	WER	30_ER	40_ER	50_ER
0.1	16.72	8	3	5
0.3	15.42	7	3	5
0.5	15.40	6	4	7
0.6	15.28	6	7	6
0.7	15.28	6	8	7
0.8	15.38	6	7	9
1.0	15.98	6	8	12
2.0	19.08	6	14	23

Table 2: Result of experiment 1. Grammar size: 500, tag sentence number: 1369

4.4 Optimal Merge Weight Exploration

In order to find the relation between the optimal merge weight and some expriment settings, we conduct experiment 2 and experiment 3.

The grammar of experiment 2 is same with experiment 1. But we use 100 low-frequency address names to tag the training corpus, and there are 165 sentences in the corpus that match the address names. The only difference between experiment 2 and experiment 1 is the number of tag sentences in corpus. The result of experiment 2 is presented in Table 3.

Comparing experiment 2 with experiment 1, we can find that when less tag sentences in the training corpus, the optimal merge weight tends to smaller. This is because smaller number of tag sentences results in lower probibility of the n-grams containing

merge weight	WER	30_ER	40_ER	50_ER
0.01	17.57	10	4	7
0.03	17.25	8	4	7
0.05	16.84	7	4	7
0.08	16.59	6	5	8
0.1	16.50	6	5	9
0.15	16.76	6	6	11

Table 3: Result of experiment 2. Grammar size: 500, tag sentences number: 165

the tag. And it's less likely for the decoder to step into the grammar WFST. From the WFST aspect, the weights of the transitions that output tag in class-based LM WFST is larger correspondingly. Obviously, smaller merge weight can compensate less tag sentences.

In experiment 3, we use 490 low-frequency address names to tag the training corpus, resulting in 1369 tag sentences. But we use a different grammar with 1280 address names, 500 address names are from the grammar in experiment 1 and remaining 780 address names are unseen in corpus. Table 4 shows the result.

merge weight	WER	30_ER	40_ER	50_ER
0.1	17.42	8	3	8
0.3	15.74	7	3	6
0.5	15.20	6	4	7
0.6	15.11	6	6	6
0.7	15.52	6	8	7
0.8	15.30	6	8	8
1.0	15.69	6	10	8
1.5	17.37	6	12	15

Table 4: Result of experiment 3. Grammar size: 1280, tag sentence number: 1369

Comparing experiment 3 with expriment 1, we can find that the number of address names in the grammar file don't influence the optimal merge weight obviously. This is because the address name number only influences the size of the grammar WFST but not the transition weight before steping into the small graph.

5 Conclusions

In this paper, we proposed a method for combining the class-based and word-based language model based on WFST, and the combination is controlled by a merge weight. Experiments shows that our approach can dramatically improve the recognition rate of the low-frequency or unseen words. In addition, we find the relation between the optimal merge weight and the experiment settings.

References

Munir Georges, Stephan Kanthak, and Dietrich Klakow. 2013. Transducer-based speech recognition with dynamic language models. In *INTERSPEECH*, pages 642–646. Citeseer.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2008. Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer.

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

Christer Samuelsson and Wolfgang Reichl. 1999. A class-based language model for large-vocabulary speech recognition extracted from part-of-speech statistics. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 1, pages 537–540. IEEE.

Johan Schalkwyk, I Lee Hetherington, and Ezra Story. 2003. Speech recognition with dynamic grammars using finite-state transducers. In *INTERSPEECH*.

Andreas Stolcke et al. 2002. Srilm-an extensible language modeling toolkit. In *INTERSPEECH*.

Wayne Ward and Sunil Issar. 1996. A class based language model for speech recognition. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 416–418. IEEE.