CSLT TECHNICAL REPORT-20150003 [Friday 30th January, 2015]

Semi-Dynamic Graph Embedding for Large Scale Language Model Adaptation

Bin Yuan[1,4], Xiaoxi Wang[1], Dong Wang[1,2*], Zhiyong Zhang[1], Xi Ma[1] and Bo Xiao[4]

[*]Correspondence: wangdong99@mails.tsinghua.edu.cn
[1]Center for Speech and Language Technology, Research Institute of Information Technology, Tsinghua University, ROOM 1-303, BLDG FIT, 100084 Beijing, China
Full list of author information is available at the end of the article

Abstract

Training a language model (LM) from scratch with a large database is very time-consuming. It is therefore attractive to adapt a well-trained LM to meet a new and specific domain. Most of modern automatic speech recognition (ASR) systems involve a static decoder that relies on a pre-compiled finite state transducer (FST), which in nature does not support LM adaptation. A possible solution embeds one or several domain-specific grammar FSTs in a domain-independent class-based n-gram LM (CNLM) FST. Current researches conduct the embedding on-the-fly in the recognition process. This dynamic embedding offers great flexibility and quick adaptation; however, it is suboptimal in terms of both accuracy and efficiency, due to the lack of graph reoptimization after embedding.

In this paper, we propose a semi-dynamic LM embedding approach, which performs the FST embedding offline and optimizes the resultant graph by standard FST operations, especially graph minimization and weight pushing. This offers a more compact and well-conditioned decoding graph than the dynamic embedding. We tested the proposed method with a large scale ASR task, and found that it can significantly improve the recognition performance on low-frequency and unseen words.

Keywords: speech recognition; language model; finite state transducer; graph embedding

1 Introduction

Automatic speech recognition (ASR) by machine has been a goal of research for decades [1]. The language model (LM) is one of the key components in ASR for providing syntactic and semantic constrains. Due to its predominant importance, modern ASR systems usually involve a strong LM that is trained with a very large text corpus, e.g., more than billions of words. The most popular language modeling approach is based on word n-grams, which predicts the probability of a word given its history by taking account of the previous n-1 words only. The training process starts from text tokenization and word normalization, and then proceeds to count the n-gram statistics, by which the n-gram LM is construed [2, 3].

A critical challenge of the current n-gram LM paradigm is that the probability estimation for low-frequency words is highly unreliable. On one hand, there are limited n-gram counts for low-frequency words in the training data, resulting in little evidence for these words in model training. On the other hand, the n-gram modeling is based on empirical probability estimation (based on n-gram counts), which itself is unreliable for low-frequency events. More seriously, the number of words of a language is generally very large and new words are invented every day. It is impossible for a LM to cover all existing words, not to even mention the words unknown at present. Taking address names as an example: most of address names are infrequent and new address names are invented every day. Ironically, addresses that are of low-frequency or new are not necessarily unimportant, particularly when we focus on a specific domain that is different from the one from which the training data is collected. We therefore encounter two different but related challenges: (1) How to give a reliable probability estimation for low-frequency words, including zero-frequency words. (2) How to adapt a well-trained LM to a specific domain for which new words need to be added and probabilities of low-frequency words need to be re-estimated.

In order to meet the first challenge, a well-known approach involves various smoothing techniques such as back-off [4] and discount [5, 6]. By this approach, a proportion of the probability mass of high-frequency words is relocated to low- or zero- frequency words, and high-order estimation falls back to low-order estimation. Another approach is to share probabilities among similar words, so that words of low-frequency can be estimated in a more reliable way by borrowing the statistical information from high-frequency words. A well-known method of this approach is the class-based n-gram LM (CNLM) [7], which groups words into classes according to certain similarity measure, and builds a CNLM by replacing the member words of a class by the tag of the class. Once the CNLM has been built, the probability of a word in the class can be computed as the product of the class probability in the CNLM and the word probability in the class.

Interestingly, the CNLM method can be used to tackle the second LM adaptation problem as well. In fact, since the probability of a word has been decomposed into two components, it is possible to adjust the class member probability according to the target domain while keeping the CNLM unchanged. It is also free to add new words into the class and assign appropriate member probabilities to them. An interesting extension of the classical CNLM is to allow complex statistical structures in the class, e.g., a stochastic grammar [8].

Most of modern ASR systems are based on finite state transducer (FST) [9]. Specifically, the n-gram LM is compiled into a word-based FST where the n-gram scores are assigned to the transitions. By the FST representation, the decoding is implemented as a simple graph search. Typically, the n-gram FST is compiled off-line, which means the decoding graph is static and does not support adaption. The idea of CNLM has led to an interesting solution for this problem, which decomposes the decoding graph into a CNLM FST which is general and static and a class FST that represents domain-specific knowledge, e.g., a list of named entities or a simple grammar. In decoding, the class FSTs are embedded in the CNLM FST, forming a domain-specific search graph. This FST embedding approach has been investigated by many researchers to offer quick domain-specific or application-specific adaptation [10, 11, 12, 13, 14, 15].

Most of the current FST embedding methods are dynamic, which means the class FSTs are embedded in the CNLM FST dynamically when performing the decoding. This offers quick on-the-fly adaptation and supports embedding large classes. However, this approach is suboptimal in terms of both efficiency and accuracy, due to the lack of graph re-optimization after embedding. Particularly, the lack of weight pushing leads to less effective beam search and hence degraded recognition performance. This paper proposes a semi-dynamic embedding approach which does not seek for on-the-fly adaptation; instead, we are interested in an off-line adaptation which is not necessarily as flexible as the dynamic embedding but produces more compact and optimized search graphs. This approach keeps the CNLM FST unchanged and modifies the class FST according to the specific domain or application. The class FST is then embedded in the CNLM FST, which is followed by several optimization steps, e.g., determinization, minimization and weight pushing. Compared to the dynamic embedding approaches, the semi-dynamic embedding does not provide realtime adaption, however the resultant search graph is more compact and optimal for beam search, leading to faster decoding and better performance. This is specifically suitable for the server-side deployment where the domain-specific knowledge needs to be updated regularly but not has to be on-the-fly, and speed and accuracy are more important than instant LM adaptation.

The remainder of this paper is structured as follows. Section 2 discusses relevant works, and our methodology is described in section 3. Section 4 presents the experiments and Section 5 concludes the paper. The embedding tool used to conduct the experiments in the paper is publicly available[1].

2 Related Work

The CNLM has been studied in many applications, such as named entity identification [16], machine translation [17], and speech recognition [7, 18, 19, 20]. Typically, the CNLM offers better performance than the word-based LM when the lexicon is huge [18, 19] or the training data is limited [7].

In [8], the conventional CNLM was extended by allowing classes to be specified by stochastic context-free grammars (SCFGs). An estimation-maximization (EM) algorithm was employed to train the model. In [10], the authors described an integrated LM where a general model linked a couple of local models. Each local model corresponded to a particular class and was trained on a subset of the training data. The models were represented in the form of weighted finite state automatons (WFSAs). Similarly, a unified model was proposed in [11], which incorporated domain-specific context-free grammars (CFGs) into a domain-independent n-gram model. This unified model can improve generalizability of the CFG and specificity of the n-gram.

More relevant works are [12, 13, 14, 15]. They are all based on FSTs. The approach proposed in [12] allowed for dynamic sub-grammars to be spliced into the main grammar on-the-fly, while preserving the dependency between neighboring words. The expansion of a sub-grammar was deferred until the corresponding sub-grammar transitions were encountered during decoding. [13] explored a unified framework that combined the benefits of grammars, n-gram LMs, and class-based LMs based on FSTs. They built a parallel model which was a weighted ‘union’ of grammars and n-gram LMs as well as a hierarchical model which was similar to the class-based LM, with the principal difference being that the classes may change their contents on-the-fly. [14] described an FST-based framework to handle dynamic vocabulary

[1] http://cslt.riit.tsinghua.edu.cn/resources.php?Public%20tools

in ASR. The proposed framework possessed advantages of low memory usage and low computation demand in vocabulary adaptation, due to the efficient composition algorithm the authors proposed and the dynamic vocabulary scheme. More recently, [15] introduced a method which embedded grammars in CNLM using a transducer-nesting technique during speech decoding.

All the above FST-based approaches concentrate on dynamic vocabularies/grammars and embed the class FSTs on-the-fly when performing decoding, and so can be called ‘dynamic embedding’. An advantage of this approach is that the class FSTs are embedded in the CNLM on demand. In other words, the class FSTs are traversed only when necessary in decoding. This avoids the expensive cost in memory usage and computation in conventional static embedding that expands all the class transitions off-line. A disadvantage associated with the on-demand embedding, however, is that the FST optimization techniques (e.g., determinization, minimization, weight pushing) cannot be applied, leading to a suboptimal search graph that is less effective in beam search. Additionally, dynamic embedding requires extra time for FST compilation, particularly if contexts are considered in the embedding.

We argue that the dynamic embedding approach is suitable for client-side applications, such as a personal voice assistant, for which the vocabulary may vary every day. However, for a service-side deployment, vocabularies and grammars do not vary that much for a particular domain, and speed and accuracy are more concerned. In this scenario, the semi-dynamic embedding proposed in this paper is more appropriate. Specifically, we follow the idea of FST-based CNLM but expand all the class transitions off-line, and optimize the expanded FST by standard FST optimization operations. To avoid the explosive cost in memory usage and computation, a context-share strategy is employed. This approach is useful for constructing domain-specific LMs or updating vocabularies for a large-scale ASR service. More details will be discussed in Section 3.

3 Semi-dynamic graph embedding

This section presents the semi-dynamic embedding approach. We start from introducing the FST architecture in speech recognition and then describe the methodology. The important design details will be highlighted.

3.1 FST in speech recognition

Most of the modern ASR systems are based on FSTs. An FST provides a graphical representation for a statistical model that maps one symbolic sequence to another, e.g., a lexicon model that maps phone sequences to word sequences. In speech recognition, all the acoustic and linguistic models can be represented by FSTs, including: the hidden Markov Model (HMM) that maps phone states to context-dependent phones (H), the decision tree that maps context-dependent phones to context-independent phones (C), the lexicon that maps context-independent phones to words (L), and the LM that assembles words into sentences (G). These four FSTs are composed to form a compositional FST that maps from low-level HMM sates to high-level sentences, i.e., $\mathcal{G} = H \circ C \circ L \circ G$.

The unified FST representation greatly simplifies the decoding algorithm, since the complicated data structures previously designed to manipulate the multiple acoustic and linguistic models and resources are not required any more. In fact, the decoding process can be implemented as a simple graph search [9, 21]. A particular advantage of the FST representation is that standard FST optimization operations, such as determinization, minimization and weight pushing can be simply employed to produce a highly compact decoding graph ($\mathcal{G}$). Especially, the weight pushing operation redistributes the LM scores so that they can be employed as early as possible. This is analog to the well-known LM look-ahead technique [22] and has been demonstrated to be rather effective for pruning unlikely paths in decoding [23].

We propose two semi-dynamic embedding approaches: the first approach, denoted by 'G embedding', embeds the class G in the CNLM G and then composes the resultant G with other components to form the decoding graph; the second approach, denoted by 'HCLG embedding', constructs the class HCLG and the CNLM HCLG respectively, and then embeds the class HCLG in the CNLM HCLG to form the decoding graph. The G embedding performs FST optimization after the embedding, while the HCLG does not perform additional optimization after the embedding, due to the constraint on memory usage and computation. Therefore, the G embedding tends to produce more 'optimal' decoding graphs, however the HCLG embedding is much more efficient and suitable for on-the-fly adaptation, which is similar to the dynamic embedding though the class HCLGs are compiled off-line.

3.2 G embedding

The G embedding approach embeds class FSTs in a CNLM FST on the G level. The class can be simply a list of domain-specific words, or a more complicated grammar. In both cases, the class can be represented by a CFG and thereof is converted to an FST. In our study, CFGs were written in the format of JSpeech Grammar Format (JSGF)[2], and we used the Sphinx toolkit[3] to convert CFGs to finite state machines (FSMs). The OpenFST toolkit[4] was then harnessed to compile an FSM to an FST (class G). For the CNLM, it can be converted to an FST by treating each n-gram history as an FST state, and the n-gram probabilities as the weights associated with the FST transitions. The Kaldi toolkit [24] was used to conduct the n-gram to FST conversion in this work.

Figure 1 exemplifies a simple CNLM G that involves just two sentences, '*like <addr> awfully*' and '*best <addr> hotel*', where '*<addr>*' is a class tag representing possible address names. Figure 2 illustrates a simple example of the class G, which involves three words ('*Paris*', '*Las Vegas*', '*London*'). Note that the class in this simple example involves a list of words, and the words are assigned the same weight. More complex grammars and more informative weight distributions are possible and more useful in practice.

When embedding the class G in the CNLM G, the class G should substitute for all the class transitions in the CNLM G. In other words, a full expansion for the CNLM G transitions labelled by the class tag. This approach, however, is extremely costly as the class transitions may occur many times in the CNLM G, leading to unaffordable memory usage and processing time. We therefore take a context-share

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

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

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

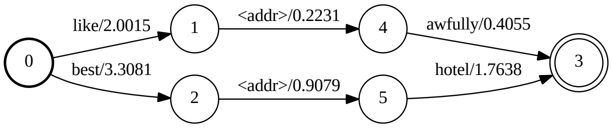

Figure 1 A simple CNLM G FST.

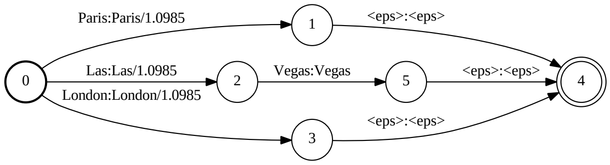

Figure 2 A simple class G FST involving three address names.

strategy that links the the leaving and entering states of a class transition to the starting and existing states of the class G, respectively. The new link-in and link-out transitions are labelled as silence, which implies that the word context that the class transition appears is ignored. An example is shown in Figure 3, which embeds the class G shown in Figure 2 in the CNLM G shown in Figure 1.

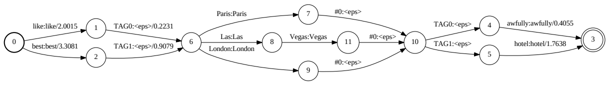

Figure 3 The G FST that embeds the class G (Figure 2) in the CNLM G (Figure 1).

For a clearer presentation, we define an FST as its starting state a, exiting state z and the transitions it involves, e.g.,

$$G = (a, z, \Xi = \{e_i;\ i = 1, 2, ..., I\}),$$

where $e_i = (s_i, t_i, v_i, l_i, r_i)$ represents the ith transition with the leaving state s_i, entering state t_i, weight v_i, input label l_i and output label r_i, and I is the total number of transitions in G. Note that we have assumed a single exiting state, which does not reduce the generability of the definition as we can always add an extra state and link all exiting states to it if there are many. The embedding algorithm is presented in Algorithm 1.

In Algorithm 1, class transitions labeled by the class tag c are searched for in the CNLM G. An important detail in the embedding process is that the the state IDs of the class G need to be mapped to the ID space of the CNLM G. This is achieved by finding the maximum state ID D^n in the CNLM G, and then add the value to all the state IDs of the class G.

The embedding weight w is used to balance the probability traversing into the class G. Note that the FSTs in Figure 1-3 are in the log semiring, which means that the weights of transitions in the FST are the logarithm of the probabilities obtained

Algorithm 1 G embedding algorithm

```
Input: CNLM G FST: $G^n = (a^n, z^n, \Xi^n)$; class G FST: $G^c = (z^c, z^c, \Xi^c)$; class tag $c$; class weight $w$.
Output: Embedded G FST: $G^m = (a^m, b^m, \Xi^m)$
  $a^m$=$a^n$; $z^m$=$z^n$; $\Xi^m$={}
  $D^n$ = max($s_i^n, v_i^n : i = 1, 2, ...I_n$) % maximum state ID in $G^n$
  for all $e_i^c$ IN $\Xi^c$ do
    $e_i^c = (D^n + s_i^c, D^n + t_i^c, v_i^c, l_i^c, r_i^c)$ % modifying the state id in $G^c$
    $\Xi^m = \Xi^m \cup \{e_i^c\}$ %add it to $G^m$
  end for
  $a^c = a^c + D^n$ % the starting state of $G^c$ $z^c = z^c + D^n$ % the exiting state of $G^c$
  k = 0
  for all $e_i^n$ IN $\Xi^n$ do
    if $l_i^n$ = c then
      ei = ($s_i^n$,$a^n$,w+$v_i^n$,TAG{k},sil) % add link-in transition
      eo = ($z^c$,$t_i^c$,0,TAG{k},sil) % add link-out transition
      $\Xi^m = \Xi^m \cup$ {ei,eo}
      for all $e_i^c$ IN $\Xi^n$ do
        e = $(D^n + s_i^c, D^n + t_i^c, v_i^c, l_i^c, r_i^c)$ % add transitions in $G^n$ to $G^m$
        $\Xi^m = \Xi^m \cup \{e\}$
      end for
      k = k + 1
    else
      $\Xi^m = \Xi^m \cup \{e_i^n\}$
    end if
  end for
```

from the CNLM. Therefore the embedding weight w added to the class transition weight of the CNLM G plays the role of a prior probability e^w for transiting into the class G. For the embedded G in Figure 3, w is set to zero so the weight from state 1 to 6 is the same as the weight from state 1 to 4 in Figure 1 (both 0.2231). Finally, the auxiliary input tags TAG{k} are used to label the new link-in and link-out transitions. These new tags guarantee that the embedded G is determinizable, according to the FST twin property [21]. These auxiliary tags are added into the lexicon and their pronunciations are set to silence.

Once the embedded G FST has been established, it is composed with other components to construct the final search graph $\mathcal{G}$, with the optimization techniques applied. The construction process is as follows:

$$\mathcal{G} = \pi_\epsilon(min(det(H \circ det(C \circ det(\widetilde{L} \circ G^m))))) \tag{1}$$

where $\widetilde{L}$ is the lexicon with the auxiliary tags added, det and min denote determinazation and minimization, respectively. π_ϵ replaces introduced auxiliary symbols by the null symbol ϵ, and removes the transitions whose input and output labels are both null. Note that Equation 1 ignores some details that are not the focus of the paper but important for the HCLG construction, such as disambiguity for homophones, auxiliary symbol augment for H and C. Readers are referred to [9] for details. It also deserves to remind that the minimization operation implicitly involves weight pushing in OpenFST, the tool used for HCLG construction in our work.

3.3 HCLG embedding

The G embedding aims to decompose the LM into a static component (CNLM) and a dynamic component (classes) so that the LM can be adapted quickly to a new domain without retraining the entire LM from scratch. The advantage of this approach is that the resultant search graph is *almost* optimized due to the optimization steps following the embedding. The disadvantage, obviously, is that it does not support on-the-fly adaptation, e.g., dynamic domain switch.

An HCLG embedding approach is presented in this section, which is similar to the dynamic embedding approaches proposed by previous researches [12, 13, 14, 15] though several differences exist. First of all, the dynamic embedding focuses on dynamic modification for the content of the class grammar, whereas the HCLG embedding proposed here does not change the class content, instead re-load new class implementations (in the form of HCLG FSTs). Secondly, the dynamic embedding expands class transitions on demand, while the HCLG embedding expands class transitions before the decoding is started. This means that the dynamic embedding requires a dynamic decoder to support the realtime transition expansion in the decoding process, however the semi-dynamic embedding does not need to change the decoder as all the embedding operations are on HCLGs and have been completed before the decoding is started.

In practice, CNLM HCLG and the class HCLG are constructed individually, following the same process formulated in Equation 1. Figure 4 illustrates a CNLM HCLG derived from a toy CNLM that involves two 3-word sentences (‘*a <addr> e*’ and ‘*i <addr> o’*) where ‘*<addr>*’ is the class tag. For clearness, the class transitions are drawn as green lines, and the leaving and entering states of these transitions are drawn as blue and red circles, respectively. We note that there are four occurrences of the class transitions in Figure 4.

Figure 5 presents the class HCLG derived from a simple grammar that involves two simple words (‘*are*’ and ‘*eye*’). Following the same idea of context-share, the class HCLG is embedded into the CNLM HCLG, by linking the leaving and entering states of the CNLM HCLG to the starting and existing states of the class HCLG, respectively. The resultant HCLG is shown in Figure 6, where the green transitions illustrate how the two HCLGs are linked as an entire search graph.

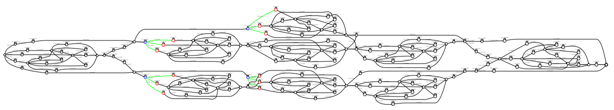

Figure 4 A simple CNLM HCLG FST.

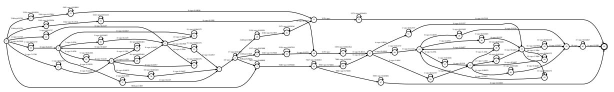

Figure 5 A simple class HCLG FST.

The HCLG embedding algorithm is similar to Algorithm 1, except that no auxiliary tags are needed to label the added link-in and link-out transitions: the link-in transitions are labeled as a silence phone, and the link-out transitions copy the labels from the original class transitions in the CNLM HCLG. Again, an embedding

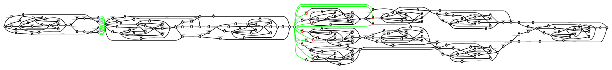

Figure 6 The HCLG FST that embeds the class HCLG in Figure 5 into the CNLM HCLG in Figure 4.

weight is used to tune the probability that the search path transiting into the class HCLG, although it is set to zero in Figure 6.

We highlight that the context-share strategy avoids the unaffordable memory and computation cost associated with the full class transition expansion. This allows very fast FST embedding and can be even done on-the-fly. In our experiments, embedding a class HCLG in a large scale CNLM HCLG derived from a 3-gram CNLM with 150k words requires less than one second on a computer with a dual-1.6GHz CPU. The disadvantage, however, is that no additional optimization (particularly weight pushing) is conducted after the embedding, possibly resulting in a suboptimal search graph.

We note that the HCLG embedding is still semi-dynamic since the compilation of the class HCLG is offline. To support on-the-fly grammar adaptation, this approach can be combined with the dynamic embedding approach that has been investigated by some researchers, e.g., [15]. This leads to the following design principle in practice: the G semi-dynamic embedding is the least flexible but the resultant graphs are the most compact and optimal, so it is suitable for domain-specific adaption on the server side; the HCLG semi-dynamic embedding is more flexible than the G embedding, but the resultant graphs are less optimal, so is suitable for user-specific model adaptation and can be employed on either the server or the client side; the dynamic embedding is the most flexible but does not support large-scale adaptation and often requires a dynamic decoder, so it is more suitable for session-specific adaptation, particularly on the client side.

4 Experiment

This section presents experiments to validate the semi-dynamic embedding approach. A large Chinese ASR task in the telecom domain was choosed to evaluate the embedding approaches, and the evaluation is in terms of the word error rate (WER) delivered by the ASR system. We first introduce the experimental configurations, and then compare the G embedding and HCLG embedding. The impact of the quality of the CNLM and the complexity of the class grammar are also investigated.

4.0.1 Database

The ASR task that we choosed aimed to transcribe conversations recorded from online service calls. The domain was telecom service and the language was Chinese. To train the acoustic model (AM), we manually transcribed 1400 hours of on-line speech recordings from a large call center service provider.

To train the LM, we collected 64 Mb text data including the transcription of the AM training speech and some logs of web-based customer service systems in the domain of telecom service. This amounted to 1.4 million sentences or 11.5 million words.

We choosed a list of NEs as the class grammar. In this work, the NEs are address names, and the selected address names are low-frequency and occur less than 10 times in the training corpus. We assume that high-frequency NEs can be well modeled by the classical 3-gram model and do not rely on embedding.

In order to evaluate the performance of the embedding approaches, a special test set was designed. It consists of 120 transcriptions and 240 utterances from 12 persons, where each transcription was recorded by 2 different persons (so 2 utterances). For each transcription, an address name is involved as a part of class grammar. The test utterances are divided into 3 subset, according to the frequency of the address name involved in the training text. The group HIGH consists of 60 utterances (30 different transcriptions) for which the address names involved are high-frequency (>10 times in the training text) and not in the grammar; the group LOW consists of 80 utterances (40 different transcriptions) whose address names are rare in the training text ($<$ 10 times) and are in the grammar; the group NONE consists of the rest 100 utterances (50 different transcriptions) whose address names do not appear in the training text but have been involved in the NE list. There are totally 10 unseen address names and each occurs 5 times in the test transcriptions. With this test set, we can compare the contribution of the embedding approaches to NEs of different frequencies.

4.0.2 Acoustic model training

The ASR system is based on the state-of-the-art HMM-DNN acoustic modeling approach, which represents the dynamic properties of speech signals using the hidden Markov model (HMM), and represents the state-dependent signal distribution by the deep neural network (DNN) model. The feature used is the 40-dimensional FBank power spectra. A 11-frame splice window is used to concatenate neighbouring frames to capture the long temporal dependency of speech signals. The linear discriminative analysis (LDA) is applied to reduce the dimension of the concatenated feature to 200.

The Kaldi toolkit was used to train the HMM and DNN models. The training process largely followed the WSJ s5 GPU recipe published with Kaldi [24]. Specifically, a pre-DNN system was first constructed based on the Gaussian mixture model (GMM), and this system was then used to produce phone alignments for the training data. The alignments were employed to train the DNN-based system.

The DNN model involves 4 hidden layers, each of which consists of 1200 units. The output layer consists of 3600 units, corresponding to the shared context-dependent phone states that inherit from the GMM-based system. The DNN training utilizes the stochastic gradient descendent (SGD) algorithm, with the objective function set to be the cross entropy between the frame targets and the DNN-based predictions. The training was conducted with a Tesla K20c GPU card, and the training process lasted roughly one week for the 1400 hours of training speech.

4.0.3 Language model training

When training the Chinese LM, the training text was first normalized, including removing unrecognized characters, unifying different encoding schemes, normalizing the spelling form of numbers and letters. After the normalization, the training data was segmented into word sequences. A word segmentation tool based on the maximum likelihood was used in this work. We selected 150,000 words as the LM lexicon, according to the word frequency in the segmented training text. The SRILM toolkit was then used to train a classical 3-gram LM, which was smoothed by the Kneser-Ney discounting.

As mentioned, the class grammar in this work was a list of NEs (address names here). When training the CNLM, the NEs were first replaced by the class tag $<addr>$, and then the CNLM was trained as a normal 3-gram LM, by treating $<addr>$ as a regular word. Note that we only replaced low-frequency NEs (<10 times), as high-frequency NEs can be well modeled by 3-grams in the CNLM.

4.1 Baseline

We take the system with the classical 3-gram LM as the baseline. Table 1 report WER of the baseline system. In order to give more insight into NE recognition, the NE error rate (NEER) is also reported, i.e., the percentage of NEs that are failed to be recognized, without considering other words. The NEER is reported for the entire test set and the three subsets: 'HIGH' involves high-frequency NEs that *not* in the class NE set; 'LOW' involves low-frequency NEs that are in the class NE set; 'NONE' involves unseen (zero-frequency) NEs that are in the class NE set.

		NEER%			
System	WER%	HIGH	LOW	NONE	TOTAL
3-gram	20.66	21.7	42.5	58.0	43.8

Table 1 WER and NEER result of the baseline.

From Table 1, we observe that low-frequency NEs are much more difficult to recognize than high-frequency NEs, and the unseen NEs are the most difficult to recognize. This is expected, as less frequent NEs tend to be less represented by the LM. Note that the unseen NEs are not totally unrecognized. For one reason, the smoothing approach allocates some probability mass to the unseen NEs when training the LM, and for the other reason, unseen NEs can be recognized in the form of a word or character sequence. Nevertheless, there is a big gap between the unseen/low-frequency NEs and the high-frequency NEs.

4.2 Graph embedding

We experimented with the semi-dynamic embedding approach. First of all, we extracted 490 low-frequency address names that occurred in the training corpus, and used these address names to tag the training text, i.e., replaced these address names by the class tag $<addr>$. There were totally 1369 sentences that were tagged. These tagged sentences, together with the original training data, were used to train the CNLM. Finally, we added 10 unseen address names to the original 490 addresses, leading to an NE set involving 500 address names.

Based on the CNLM and the NE set, we experimented with the G embedding and HCLG embedding approaches, with different settings of the embedding weight w. The experiments were conducted following the same process as the baseline system. Note the 'LOW' test NE subset is included in the 490 NE list. The results of the G embedding and HCLG embedding are presented in Table 2 and Table 3, respectively.

System	w	WER %	NEER%			
			HIGH	LOW	NONE	TOTAL
3-gram	-	20.66	21.7	42.5	58.0	43.8
G Embedding	1	15.72	21.7	21.2	16.0	19.2
	2	14.84	21.7	17.5	10.0	15.4
	3	15.35	21.7	16.3	10.0	15.0
	4	15.23	21.7	16.3	10.0	15.0
	5	15.33	25.0	12.5	7.0	13.3
	6	16.57	31.7	10.0	7.0	14.2

Table 2 WER and NEER result with G embedding. NE set size: 500, tagged sentences: 1369.

System	w	WER %	NEER%			
			HIGH	LOW	NONE	TOTAL
3-gram	-	20.66	21.7	42.5	58.0	43.8
HCLG Embedding	2	17.35	23.3	33.8	33.0	30.8
	3	15.94	23.3	22.5	28.0	25.0
	4	15.25	25.0	20.0	20.0	21.2
	5	14.86	25.0	20.0	13.0	18.4
	6	16.08	26.7	15.0	14.0	17.4

Table 3 Result with HCLG embedding. NE set size: 500, tagged sentences: 1369.

From the results we can see that both the G embedding and the HCLG embedding can considerably improve performance of the ASR system, in terms of both WER and NEER. For both the two approaches, a large weight tends to provide better recognition for low-frequency and unseen NEs but lead to more errors for high-frequency words. This is expected since a large weight encourages traveling into the class graph in decoding and thus recognize more NEs in the class grammar. Interestingly, a reasonable w does not lead to performance reduction on high-frequency NEs, which means that the embedding approaches are safe in general.

Comparing the two embedding approaches, we see slightly better performance with the G embedding, if we compare the best performance with an optimal embedding weight (14.84 with G embedding v.s. 14.86 with HCLG embedding). However the G embedding seems more reliable, as the improvement on the low-frequency NEs does not impact the performance on the high-frequency NEs, if the embedding weight is moderately set. Additionally, the G embedding looks more effective than the HCLG embedding in boosting low-frequency and unseen NEs. This advantage with the G embedding, as we discussed, can be attributed to the re-optimization after the embedding. Nevertheless, the HCLG embedding works fairly well, and it possesses the advantage of more flexibility.

As a summary, the proposed semi-dynamic approach can safely improve ASR performance on low-frequency and unseen NEs, and does not impact the performance on other words if a moderate embedding weight is set. This lends itself to quick domain-specific adaptation where the domain-specific words and phrases that need to recognize are often of low-frequency or unseen in the original model.

4.3 Impact of CNLM quality

The quality of the embedded graph is impacted by the quality of the CNLM and the complexity of the NE set. This section studies the impact of the CNLM quality. We used the G embedding to conduct the study, and the experimental settings were the same as in the previous section, except that the number of tagged sentences was reduced. To achieve that, we choosed 100 low-frequency address names to tag the training corpus, instead of 490 address names in the previous experiment. There were 165 sentences that were tagged, compared to the 1369 sentences in the previous experiment. The less tagged sentences led to a weaker CNLM for the class tags. The results are shown in Table 4.

System	w	WER %	NEER%			
			HIGH	LOW	NONE	TOTAL
3-gram	-	20.66	21.7	42.5	58.0	43.8
G embedding	2	16.25	23.3	27.5	23	24.6
	3	15.45	23.3	22.5	20	21.7
	4	15.59	23.3	21.2	15	19.2
	5	15.62	21.7	20.0	13	17.5
	6	15.50	21.7	17.5	12	16.3
	7	15.59	23.3	21.2	15	19.2
	8	15.62	28.3	20.0	13	19.2

Table 4 Result with G embedding. NE set size: 500, tagged sentences: 165

It can be seen that the lower quality of the CNLM does impact the performance, but the impact seems marginal (WER increases from 14.84% to 15.50%). This suggests that a moderate number of tagged sentences are likely sufficient to train a reasonable CNLM. Another observation is that the optimal embedding weight increases from 2.0 to 6.0. This is not surprising as less tagged sentences leads to a lower probability for transiting into the class graphs, and therefore requires a larger embedding weight to compensate for the lost probability.

4.4 Impact of NE set

The final experiment investigated the impact of complexity of the NE set on the quality of the embedded graph. Recall that the NE set involved 500 address names in the previous experiments. In this experiment, the NE set was augmented with another 780 unseen addresses, leading to an NE set involving 1280 address names. Due to the large number of members, the complexity of the class was significantly increased. The same 490 NEs were used to tag the training text, so the number of tagged sentences remained 1369. Again, we focused on the G embedding. Table 4 shows the results.

System	w	WER %	NEER%			
			HIGH	LOW	NONE	TOTAL
3-gram	-	20.66	21.7	42.5	58.0	43.8
G embedding	0	15.96	23.3	23.7	19	21.7
	1	15.40	21.7	21.2	16	19.2
	2	15.08	20.0	17.5	10	15.0
	3	15.33	20.0	16.3	10	14.6
	4	15.20	25.0	16.3	10	15.8
	5	15.47	25.0	12.5	7	13.3
	6	16.76	31.7	10.0	7	14.2

Table 5 Result with G embedding. NE set size: 1280, tagged sentences: 1369

We see that the WER with the more complex NE set increases from 14.84% to 15.08%. This is expected and reasonable, considering that much more unseen address names are added into the NE set. Fortunately, the performance degradation seems not that much; particularly, including more unseen NEs does not impact the performance on frequent NEs. This is a desirable property and it suggests that involving a relatively large set of unseen NEs is possible with the embedding approach. Another finding is that the optimal embedding weight remains the same with the large NE set. This suggests that the embedding weight is largely determined by the amount of tagged sentences in CNLM training and has not to be re-tuned if the NE set is changed. This is also a good property in practical usage.

5 Conclusions

In this paper, we propose two semi-dynamic embedding approaches that combine a large-scale CNLM that is domain-independent and trained with a large amount of data and some class grammars that are domain-dependent and can be quickly adapted. The decoding graph with the G embedding is more compact and optimal while the HCLG embedding possesses the advantage of more flexible adaptation. The experiments demonstrated that the proposed two embedding approaches can significantly improve the performance of speech recognition on low-frequency and unseen NEs, especially with the G embedding. This lends the semi-dynamic approach to quick domain-specific adaptation for which many words and phrases are low-frequency or unseen in the original model. In addition, we found that the impact of the CNLM quality and the complexity of the class grammars is moderate, and the embedding weight is rather robust against the change of grammars.

Acknowledgement

This research was supported by the National Science Foundation of China (NSFC) under the project No. 61371136, and the MESTDC PhD Foundation Project No. 20130002120011. It was also supported by Sinovoice and Huilan Ltd.

Author details

[1]Center for Speech and Language Technology, Research Institute of Information Technology, Tsinghua University, ROOM 1-303, BLDG FIT, 100084 Beijing, China. [2]Center for Speech and Language Technologies, Division of Technical Innovation and Development, Tsinghua National Laboratory for Information Science and Technology, ROOM 1-303, BLDG FIT, 100084 Beijing, China. [3]Department of Computer Science and Technology, Tsinghua University, ROOM 1-303, BLDG FIT, 100084 Beijing, China. [4]Beijing University of Posts and Telecommunications, No 10, Xitucheng Road, Haidian District, 100876 Beijing, China.

References

1. Lawrence R Rabiner and Biing-Hwang Juang, *Fundamentals of speech recognition*, vol. 14 of *Signal processing*, PTR Prentice Hall Englewood Cliffs, 1993.
2. Roni Rosenfield, "Two decades of statistical language modeling: Where do we go from here?," *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1270–1278, 2000.
3. Jerome R Bellegarda, "Statistical language model adaptation: review and perspectives," *Speech Communication*, vol. 42, no. 1, pp. 93–108, 2004.
4. Slava Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 35, no. 3, pp. 400–401, 1987.
5. Reinhard Kneser and Hermann Ney, "Improved backing-off for m-gram language modeling," in *Proceedings of ICASSP*, 1995, pp. 181–184.
6. Stanley F Chen and Joshua Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–393, 1999.
7. Wayne Ward and Sunil Issar, "A class based language model for speech recognition," in *Proceedings of ICASSP*, 1996, pp. 416–418.
8. John Gillett and Wayne Ward, "A language model combining trigrams and context-free grammars," in *Proceedings of ICSLP*, 1998, pp. 2319–2322.
9. Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.
10. Alexis Nasr, Yannick Esteve, Frédéric Béchet, Thierry Spriet, and Renato De Mori, "A language model combining n-grams and stochastic finite state automata," in *Proceedings of EUROSPEECH*, 1999, pp. 2175–2178.
11. Ye-Yi Wang, Milind Mahajan, and Xuedong Huang, "A unified context-free grammar and n-gram model for spoken language processing," in *Proceedings of ICASSP*, 2000, pp. 1639–1642.
12. Johan Schalkwyk, I Lee Hetherington, and Ezra Story, "Speech recognition with dynamic grammars using finite-state transducers," in *Proceedings of INTERSPEECH*, 2003, p. 1969–1972.
13. Hans JGA Dolfing, Pierce Gerard Buckley, and David Horowitz, "Unified language modeling using finite-state transducers with first applications," in *Proceedings of INTERSPEECH*, 2004.
14. Paul R Dixon, Chiori Hori, and Hideki Kashioka, "A specialized wfst approach for class models and dynamic vocabulary," in *Proceedings of INTERSPEECH*, 2012.

15. Munir Georges, Stephan Kanthak, and Dietrich Klakow, "Transducer-based speech recognition with dynamic language models.," in *Proceedings of INTERSPEECH*, 2013, pp. 642–646.
16. Jian Sun, Jianfeng Gao, Lei Zhang, Ming Zhou, and Changning Huang, "Chinese named entity identification using class-based language model," in *Proceedings of CoLING*, 2002, pp. 1–7.
17. Jakob Uszkoreit and Thorsten Brants, "Distributed word clustering for large scale class-based language modeling in machine translation," in *Proceedings of ACL*, 2008, pp. 755–762.
18. Christer Samuelsson and Wolfgang Reichl, "A class-based language model for large-vocabulary speech recognition extracted from part-of-speech statistics," in *Proceedings of ICASSP*, 1999, pp. 537–540.
19. Edward WD Whittaker and Philip C Woodland, "Efficient class-based language modelling for very large vocabularies," in *Proceedings of ICASSP*, 2001, pp. 545–548.
20. Imed Zitouni, Olivier Siohan, and Chin-Hui Lee, "Hierarchical class n-gram language models: towards better estimation of unseen events in speech recognition," in *Proceedings of INTERSPEECH*, 2003, pp. 237–240.
21. Mehryar Mohri, Fernando Pereira, and Michael Riley, "Speech recognition with weighted finite-state transducers," in *Springer Handbook of Speech Processing*, pp. 559–584. Springer, 2008.
22. Stefan Ortmanns, Hermann Ney, and Andreas Eiden, "Language-model look-ahead for large vocabulary speech recognition," in *Proceedings of ICSLP*, 1996, pp. 2095–2098.
23. Mehryar Mohri and Michael Riley, "A weight pushing algorithm for large vocabulary speech recognition," in *Proceedings of European Conf. on Speech Communication and Technology*, 2001, pp. 1603–1606.
24. Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely, "The kaldi speech recognition toolkit," in *Proceedings of ASRU*, 2011, pp. 1–4.