INTERSPEECH 2017
August 20–24, 2017, Stockholm, Sweden

A Rescoring Approach for Keyword Search Using Lattice Context Information

Zhipeng Chen, Ji Wu

Multimedia Signal and Intelligent Information Processing Lab,
Department of Electronic Engineering, Tsinghua University, Beijing, China

`zp-chen11@mails.tsinghua.edu.cn, wuji_ee@tsinghua.edu.cn`

Abstract

In this paper we present a rescoring approach for keyword search (KWS) based on neural networks (NN). This approach exploits only the lattice context in a detected time interval instead of its corresponding audio. The most informative arcs in lattice context are selected and represented as a matrix, where words on arcs are represented in an embedding space with respect to their pronunciations. Then convolutional neural networks (CNNs) are employed to capture distinctive features from this matrix. A rescoring model is trained to minimize term-weighted sigmoid cross entropy so as to match the evaluation metric. Experiments on single-word queries show that lattice context brings complementary gains over normalized posterior scores. Performance on both in-vocabulary (IV) and out-of-vocabulary (OOV) queries are improved by combining NN-based scores with standard posterior scores.

Index Terms: keyword search, rescore, neural networks, speech recognition

1. Introduction

Keyword search (KWS), also known as spoken term detection (STD), is a task whose goal is to find all occurrences of a keyword in speech data. In this task, a keyword/query is given as a sequence of orthographic terms. LVCSR often plays an important role in bridging the gap between textual queries and audio data.

A typical KWS system is based on indices generated from ASR results (e.g., lattices). A hit in indices is scored by the posterior score of the keyword given the lattice and is often normalized with a keyword-specific thresholding method. Hits with scores above thresholds are returned by the system. Therefore, the performance of a keyword search system is heavily dependent on the quality of posterior scores in lattices. For a low-resource language, it is difficult to train a high performance ASR system to provide high quality posterior scores. In the past few years, some researchers paid their attention to rescoring/re-ranking hypothesized detections by involving more information or classifying with a model trained on tuning set. In [1], the authors proposed a graph-based re-ranking approach considering acoustic similarity between two hits. In [2], a keyword verification method was proposed. However, these methods require revisiting the acoustic data in the testing phase, which is the test audio reuse (TAR) condition defined in Babel program [3]. For the no test audio reuse (NTAR) case, some approaches based on machine learning were investigated [4, 5], but feature engineering is critical in most of these methods.

This work focuses on improving KWS performance in NTAR condition without laborious feature engineering by taking advantage of the capacity of neural networks in automatic feature extraction. A basic assumption is that the context of a detected region provides complementary information which may be helpful to keyword search. This can be learnt from a held-out tuning set. To avoid test audio reuse, we extract context information from lattices rather than audio data. This is similar to the support vector machine (SVM) based approach in [6], but differs in several aspects. First, the SVM-based method requires training an SVM for every hit list, but our model is trained using tuning set without knowledge of concerned keywords. Second, the dimension of word representation in that work depends on a pre-defined vocabulary. Instead, our work represents words in a dense embedding space. Third, acoustic features are exploited in that work, while our work pays attention to NTAR condition. So far we consider only single-word queries because KWS performance on these queries is generally worse than that on multi-word queries. But this approach is possibly flexible enough to be applied on multi-word queries.

The rest of this paper is structured as 4 sections. Section 2 introduces a classical pipeline of KWS systems. Our proposed approach is presented in Section 3. Then experimental results are described in Section 4. We conclude in Section 5.

This work was supported by the National Natural Science Funds of China under Grant 61571266.

2. KWS pipeline

The indices for spoken keyword search are usually created on lattices/confusion networks [7]. The estimated posterior probability of a word in a time interval given the whole lattice is commonly used as an initial hit score, which is then normalized [8, 9] to make scores among keywords comparable. In this work, we use a popular keyword-specific thresholding method to do score normalization [8, 10]. To handle OOV keywords, an approach using IV words as proxies is applied [11]. Our goal is to refine posterior scores of hits with complementary information from lattice context.

3. Proposed rescoring model

An overview of our rescoring model is illustrated in Figure 1. It is a neural network with a single sigmoidal unit in the output layer performing binary classification. Given a hypothesized detection as input, the model outputs an NN-based score between 0 and 1. A hit contains a keyword, a retrieved utterance with a time interval inside, and a posterior score. In this section, we will first describe how hits are represented as the input of neural networks, and then introduce the structure of proposed rescoring model and the corresponding loss function for training.

3.1. Word pronunciation embedding

The keyword and words on lattice arcs are probably the most critical information of a hit. Obviously, using one-hot representation would cause data sparsity issue. Inspired by the widely

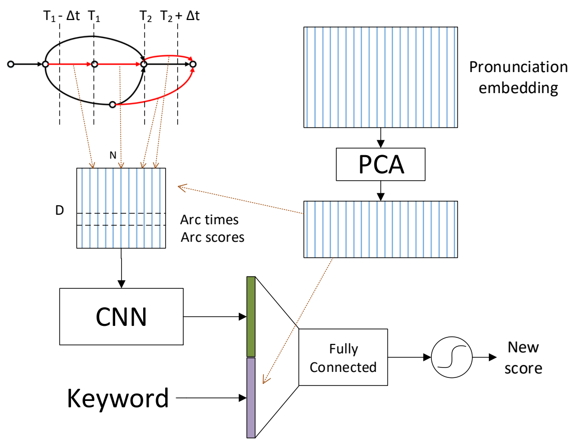

Figure 1: *Schema of proposed rescoring approach.*

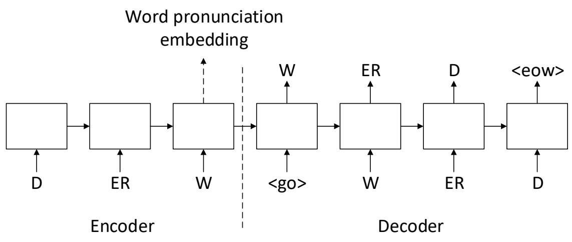

Figure 2: *Sequence-to-sequence auto-encoder for word pronunciation embedding. Decoding starts with a special input symbol $\langle go \rangle$ and ends with an end-of-word output symbol $\langle eow \rangle$.*

used word embedding in natural language processing tasks [12] and several works about acoustic embedding [13, 14], we decided to give each word a distributed representation so that words with similar pronunciations are expected to be close in an embedding space.

The model for representing pronunciations is an sequence-to-sequence auto-encoder based on a recurrent neural network (RNN) with long short-term memory (LSTM) units, which is similar to neural machine translation [15], sequence-to-sequence grapheme-to-phoneme conversion [16] and audio word2vec [14]. First, each word is converted to a sequence of phonemes according to the pronunciation lexicon. Then a sequence-to-sequence auto-encoder is trained on IV words. After training, we can extract the final state of the encoder to represent any word if we know its pronunciation. OOV words can also be represented once their pronunciations are obtained through grapheme-to-phoneme (G2P) conversion, which is crucial when handling OOV keywords.

Figure 2 depicts the structure of this auto-encoder. The encoder accepts a time-reversed sequence of phonemes as in [17] and converts them to vectors in a phoneme embedding space. After encoding, the final state then serves as the initial state of the decoder. During training, input phonemes of the decoder are almost the same as those of the encoder except for the first symbol $\langle go \rangle$. Notice that the final state of the encoder (i.e., the embedding vector) is the only input to the decoder, so it is expected to memorize the whole pronunciation.

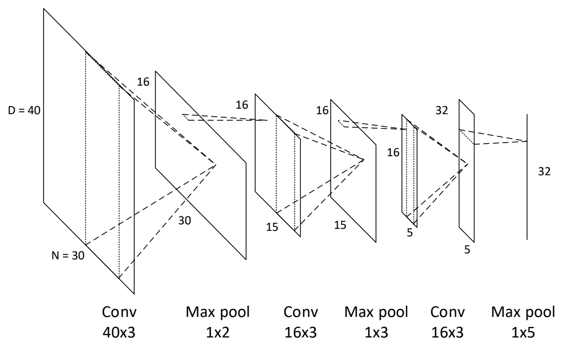

Figure 3: *CNN structure when $D = 40$ and $N = 30$. Each column of the input matrix represents a selected arc in lattice context. The numbers of feature maps are 16,16,16,16 and 32. The final max pooling layer outputs a 32-dim vector representing lattice context.*

3.2. Representation of context in lattices

For each hit, we first extract a sub-lattice on the time interval (extended by 0.5 seconds on either side) in the following way:

- On the full lattice of the target utterance, do weight pushing to make the sum of each state's outgoing transition probabilities is equal to 1, then compute the forward probability α for each state.
- Extract every arc whose time-span overlaps with the target time interval.

These extracted arcs are then merged as follows:

- Compute the posterior score for each arc.
- Cluster arcs with the same word and overlapping time-spans [7].
- In each cluster, merge arcs to a single arc which inherits almost all properties from the top-scored arc, except that the merged posterior score is the sum of all arcs' scores.

We try to select N most informative arcs in this sub-lattice. To ensure that central arc is ranked around $\frac{N}{2}$, we choose top-scored $\frac{N}{2}$ arcs from the first half time interval and the other $\frac{N}{2}$ from the second half. Then all selected arcs are sorted by midpoints of their time-spans.

Each arc is represented by a D-dimensional vector consisting of the mentioned word pronunciation embedding vector, begin time, end time and 6 arc-related scores. These scores include posterior scores of arc words, forward probabilities of both previous states and next states, language model scores, acoustic model scores and their average per frame. Thus, the context of a hit is represented as a $D \times N$ matrix.

3.3. Rescoring model

Our model first transforms the sub-lattice matrix into a compact vector. This vector and the keyword embedding vector are then concatenated and fed into several fully-connected layers to get the final output. It is worth mentioning that the original posterior score of the keyword is not directly passed to this model due to unsatisfactory results. A possible explanation is that keyword posterior scores are too instructive to help the model learn something complementary. Learning possibly tends to converge to a local minimum close to the original scores, thus losing complementarity.

Although the representation of a sub-lattice is designed to be placed in a relatively reasonable arc order, there still exist variations in the position of distinctive features. Since the amount of tuning data is strictly limited, it is difficult to get sufficient hits to cover these variations. The convolutional neural network (CNN) is a powerful structure to handle small shifts of distinctive features and has been applied successfully in image, speech and natural language processing. We utilize CNNs with rectified linear (ReLU) units in the first few layers of our rescoring model, accepting the $D \times N$ matrix as input. Similar to its application in sentence classification [18], the height of each filter is equal to D since spatially local correlation is meaningless across the dimensions of word pronunciation embedding. The structure of our CNNs is shown in Figure 3.

3.4. Loss function

In general, an effective classifier requires that examples in both training set and test set follow the same distribution and all examples make an equal contribution to the loss function. However, this is not the case in KWS. The most popular evaluation metric in KWS is term weighted value (TWV).

$$TWV(\theta) = \frac{1}{|Q|} \sum_{q \in Q} \left[\frac{N_{Recall}(q,\theta)}{N_{True}(q)} - \frac{\beta N_{FA}(q,\theta)}{T - N_{True}(q)} \right], \tag{1}$$

where Q is the set of keywords, θ is the decision threshold. It is a metric based on recall rate and penalized by false alarm rate. Performance of each keyword contributes equally to the overall performance without consideration of the number of its examples. Several loss functions have been proposed to address this issue [4, 5]. Similar to the logistic lower-bound of TWV in [4], we use a " term-weighted sigmoid cross entropy " as follows to treat examples unequally in training phase:

$$\begin{aligned} L(y, \hat{y}) = \frac{1}{|Q|} \sum_{q \in Q} \sum_{i \in H(q)} \Big[& -\frac{1}{N_{True}(q)} y_i \log \hat{y}_i \\ & - \frac{\beta}{T - N_{True}(q)} (1 - y_i) \log(1 - \hat{y}_i) \Big], \end{aligned} \tag{2}$$

where $H(q)$ is the the hit list of keyword q, y refers to labels and $\hat{y}$ refers to model outputs.

4. Experiments

4.1. Experimental setup

Experiments were conducted on Swahili conversational telephone speech collection [1] from IARPA Babel Program [3]. Our approach was evaluated in Very Limited Language Pack (VLLP) condition with only 3 hours transcribed speech data for acoustic model and language model training and web text was not used. We applied the graphemic lexicon in [19], which, in Swahili case, is almost treating grapheme sequences as phoneme sequences. The acoustic model was trained by fine tuning on a multilingual DNN seed model with 3 hours transcribed data. Kneser-Ney 3-gram language model was trained with 3 hours transcriptions.

[1]This effort uses IARPA Babel Program Swahili language collection release IARPA-babel202b-v1.0d (VLLP).

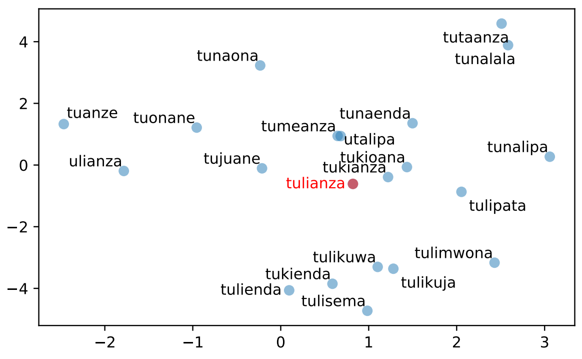

Figure 4: *20 closest neighbors of Swahili word "tulianza" in the pronunciation embedding space.*

Besides the training set for speech recognition, there are two more data sets: a 3 hours tuning set (tun) and a 10 hours development set (dev). The development set serves as the target speech collection of KWS on which we report experimental results. The tuning set is used to provide training data for our rescoring model. Besides, we also use it to tune hyper-parameters of pronunciation auto-encoder. Word error rates (WERs) of tun and dev are 62.5% and 60.5% respectively.

4.2. Results and discussion

4.2.1. Pronunciation embedding results

All IV words (5130 in total) were utilized to train the pronunciation auto-encoder. We investigated structures of the pronunciation auto-encoder and evaluated them on OOVs in tuning set (3066 in total). Table 1 presents phoneme error rate (PER) and word error rate on tuning set OOVs. A proper configuration is 1 LSTM layer, 128 dimensional LSTM cell and 20 dimension for embedding input symbols (phonemes). In this configuration, PER and WER of development set OOVs (6171 in total) are 3.0% and 9.7% respectively. Figure 4 shows neighbors of word "*tulianza*" by projecting embedding vectors to a 2D space using principle component analysis (PCA). We can see that these words are quite similar in terms of grapheme sequences.

Since the size of embedding vectors affects the structure of our rescoring model, for convenience, we integrate pronunciation embedding into the rescoring model by reducing dimensionality with PCA.

Table 1: *Results of pronunciation auto-encoder on OOVs in tun.*

#LSTM layers	cell size	phoneme embedding size	PER (%)	WER (%)
1	128	20	**3.1**	**9.5**
1	128	30	3.3	10.2
1	256	20	3.2	10.8
2	128	20	4.4	13.8

4.2.2. KWS results

The keyword list for evaluating KWS on the development set is the official one in NIST OpenKWS15, excluding no occurrence keywords and multi-word keywords. To evaluate our proposed model, we created several sets of hits, as listed in

Table 2. In order to train the rescoring model offline, we created a new keyword list with VLLP vocabulary ($VLLP\text{-}IV$) for obtaining training hits but excluded words with fewer than three phonemes to avoid too many false alarms. OOV words in the tuning set constitute another keyword list called $VLLP\text{-}OOV$. Standard KWS pipeline is performed for both $VLLP\text{-}IV$ and $VLLP\text{-}OOV$ keyword lists on tuning set. Then we obtain hit lists of these queries. As the weights of false alarms in Equation 2 are relatively small, we restricted the number of negative examples not greater than 3 times the number of positive examples for each hit list. Furthermore, hits with normalized scores less than 0.1 are discarded for both training and test set. This filtering can speed up training without hurting KWS performance. We also put totally missing true hits (i.e., not appearing in lattices) into training sets as positive examples. After that, the proportion of positive hits in both training sets is around 50%.

Table 2: *Summary of datasets and baseline performance in Maximal TWVs (MTWVs).*

Data set	Keyword list (size)	# Hits	MTWV
tun-IV	VLLP-IV (1920)	23.5k	-
tun-OOV	VLLP-OOV (3052)	11.0k	-
dev-IV	NIST-IV (345)	11.1k	0.3163
dev-OOV	NIST-OOV (462)	11.7k	0.0507

The rescoring model is optimized with shuffled mini-batch of 64 hits and Adam gradient descent [20]. L2 regularization and 50% dropout are applied to fully connected layers to alleviate overfitting. In training, we randomly choose a certain proportion of utterances and use all their hits as a validation set. Validation loss is evaluated every 200 steps (i.e., 200 mini-batches). The model with the lowest validation loss is selected.

Model configuration was explored using $tun\text{-}IV$ as training set and $dev\text{-}IV$ as test set. First, we compare the difference between standard sigmoid cross entropy and the term-weighted one. Table 3 presents Oracle TWVs (OTWVs) computed directly by output scores of our model. It is obvious that term-weighted loss outperforms standard loss, since it matches the way TWV is computed. Moreover, using term-weighted loss takes much fewer steps to reach the best model for validation set. However, we can also see that these results are much worse than those of baseline. It is not surprising for the reason that we didn't involve posterior scores directly in our rescoring approach. We hope that new scores contain complementary information to posterior scores. To exploit this information, we first normalize new scores in the same way as in Section 2, and then combine them with normalized posterior scores simply by computing their weighted average as the final score. The weight of posterior scores is scanned from 0.0 to 1.0 with a step of 0.1. Our experience suggests that the oracle weight is 0.5 or 0.6 in most cases. The remaining results are reported in oracle MTWVs in this sense (referred to as OMTWV).

Table 3: *Comparison between standard and term-weighted sigmoid cross entropy criterion. Steps here means the number of steps required to reach the lowest validation loss.*

Loss	raw OTWV	steps
standard	0.1827	18k
term-weighted	**0.2002**	**6k**

Next, we show results on various settings to find a proper one and to validate the effectiveness of pronunciation embedding and arc features (times and scores). Most of these experiments use a single fully-connected hidden layer with 64 nodes to accept the embedding vector of a query and the output vector from CNNs. As presented in Table 4, setting pronunciation embedding size to 32 obtains the best OMTWV. However, randomly setting embedding vectors or discarding time and score features degrades the performance, which indicates that CNNs have learnt something complementary from sub-lattice matrices. Adding one more fully-connected hidden layer leads to a further improvement.

Table 4: *Results of different embedding size / features / fully-connected (FC) structures.*

FC	Embedding size	OMTWV	Impr. (%)
	no rescore	0.3163	-
64	32 (random)	0.3291	4.1
	32 (no times or scores)	0.3277	3.6
	16	0.3315	4.8
	32	**0.3380**	**6.9**
	64	0.3347	5.8
64-64	32	**0.3424**	**8.3**

At last, we apply this approach to OOV hits. For OOV keywords, hits are obtained through proxies, which may lead to a lattice context quite different from IVs'. A more reasonable way is to train a new model with proxy-based hits. Therefore, we trained another rescoring model with an identical structure but used $tun\text{-}OOV$ as training set. Results are presented in Table 5, showing that a more matched training set results in a better performance.

Table 5: *Performance on dev-OOV using different training sets.*

Training set	OMTWV	Impr. (%)
no rescore	0.0507	-
tun-IV	0.0540	6.6
tun-OOV	**0.0547**	**7.9**

5. Conclusion

We propose an NN-based rescoring approach for spoken keyword search in NTAR condition, taking advantage of context information in lattices. As part of it, a sequence-to-sequence auto-encoder is used to embed word pronunciations in a dense space. Then lattice context is represented as a sub-lattice matrix, from which CNNs help extracting distinctive features. These features, together with the embedding vector of a keyword, are fed into several fully-connected layers and a single final sigmoid unit to perform binary classification. By combining these output scores with normalized posterior scores, MTWVs are improved by 8.3% and 7.9% for IV and OOV keywords, respectively. Compared with other machine learning approaches, this approach satisfies NTAR condition and requires much less feature engineering. It is possibly extensible to multi-word keywords and subword-based systems, which is also the direction of our future work.

6. References

[1] H.-y. Lee, Y. Zhang, E. Chuangsuwanich, and J. R. Glass, "Graph-based re-ranking using acoustic feature similarity between search results for spoken term detection on low-resource languages," in *Proceedings of INTERSPEECH*, 2014, pp. 2479–2483.

[2] Z. Lv, M. Cai, W.-Q. Zhang, and J. Liu, "A novel discriminative score calibration method for keyword search," in *Proceedings of INTERSPEECH*, 2016, pp. 745–749.

[3] "IARPA Babel program." [Online]. Available: http://www.iarpa.gov/index.php/research-programs/babel

[4] K. Audhkhasi, A. Sethy, B. Ramabhadran, and S. S. Narayanan, "Semi-supervised term-weighted value rescoring for keyword search," in *Proceedings of ICASSP*. IEEE, 2014, pp. 7869–7873.

[5] V. Soto, L. Mangu, A. Rosenberg, and J. Hirschberg, "A comparison of multiple methods for rescoring keyword search lists for low resource languages." in *Interspeech*, 2014, pp. 2464–2468.

[6] T.-W. Tu, H.-Y. Lee, and L.-S. Lee, "Improved spoken term detection using support vector machines with acoustic and context features from pseudo-relevance feedback," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 383–388.

[7] D. Can and M. Saraclar, "Lattice indexing for spoken term detection," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 8, pp. 2338–2347, 2011.

[8] D. R. Miller, M. Kleber, C.-L. Kao, O. Kimball, T. Colthurst, S. A. Lowe, R. M. Schwartz, and H. Gish, "Rapid and accurate spoken term detection." in *Proceedings of INTERSPEECH*, 2007, pp. 314–317.

[9] D. Karakos, R. Schwartz, S. Tsakalidis, L. Zhang, S. Ranjan, T. Ng, R. Hsiao, G. Saikumar, I. Bulyko, L. Nguyen *et al.*, "S-core normalization and system combination for improved keyword spotting," in *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2013, pp. 210–215.

[10] Z. Chen, Z. He, P. Lv, and J. Wu, "Improving keyword search by query expansion in a probabilistic framework," in *9th International Symposium on Chinese Spoken Language Processing (ISCSLP)*. IEEE, 2014, pp. 187–191.

[11] G. Chen, O. Yilmaz, J. Trmal, D. Povey, and S. Khudanpur, "Using proxies for OOV keywords in the keyword search task," in *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2013, pp. 416–421.

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[13] S. Bengio and G. Heigold, "Word embeddings for speech recognition." in *Proceedings of INTERSPEECH*, 2014, pp. 1053–1057.

[14] Y.-A. Chung, C.-C. Wu, C.-H. Shen, H.-Y. Lee, and L.-S. Lee, "Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder," *Proceedings of INTERSPEECH*, pp. 765–769, 2016.

[15] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[16] K. Yao and G. Zweig, "Sequence-to-sequence neural net models for grapheme-to-phoneme conversion," *arXiv preprint arXiv:1506.00196*, 2015.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[18] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[19] M. Gales, K. Knill, and A. Ragni, "Unicode-based graphemic systems for limited resource languages," in *Proceedings of ICASSP*. IEEE, 2015, pp. 5186–5190.

[20] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.