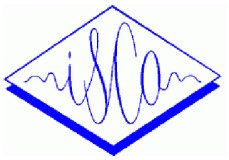

INTERSPEECH 2012
ISCA's 13th Annual Conference
Portland, OR, USA
September 9-13, 2012

Improvements in Japanese Voice Search

Ken-ichi Iso[1], *Edward Whittaker*[2], *Tadashi Emori*[1], *Jumpei Miyake*[1]

[1]Yahoo Japan Corporation, Tokyo, Japan
[2]Inferret Limited, Northampton, England

kiso@yahoo-corp.jp, ed@inferret.co.uk, taemori@yahoo-corp.jp, jmiyake@yahoo-corp.jp

Abstract

This paper describes work on Japanese voice-search at Yahoo! Japan. We first describe several implementation details of our WFST-based internal decoder which make the voice-search task more efficient including a simple, but effective, compressed WFST arc representation. This permits a ˜2Gb memory decoder process for a 1 million word vocabulary and 35 million N-gram language model. We then describe our baseline system using the decoder and compare it against two open-source decoders, Juicer and Julius. We also describe our initial attempts to adapt the baseline system through simple language model adaptation using manually transcribed anonymized voice queries. To achieve this we present a sequence of WFST operations which preserve consistency of segmentation between manual and automatic transcriptions. We show that even using this simple adaptation method we obtain a relative reduction of up to 4.6% in sentence error rate and 8.2% in character error rate.

Index Terms: ASR, Japanese, voice search, WFST

1. Introduction

Over the past few years voice-input for web-search has become both a hot research topic as well as a practical input method for many users to query web search engines from their mobile devices[6, 12]. The voice-search task is distinguished from other tasks by the vast amounts of data that are typically available for language model (LM) training and the large vocabularies required to achieve low out-of-vocabulary (OOV) rates. Much has been written to date about research performed on English language voice search[2, 3, 4] but comparatively little regarding other languages[5].

This paper describes initial development work and experiments on Japanese web voice-search that have been performed at Yahoo! Japan. We explain the salient characteristics and implementation details of our decoder and then describe experimental results on samples of anonymized real-world user data, collected via an iOS application. We compare the performance in terms of sentence error rate (SER) and real-time factor (RTF) of two open-source speech decoders against our internal decoder. While our decoder uses 76% less memory than a comparable open-source decoder, we also show that despite the significant requirements of the task good performance can also be achieved using open source software.

We also look at some of the characteristics of the Japanese voice search task, including issues of word segmentation, multiple orthographies and scoring. In addition, we describe a series of simple LM adaptation experiments to improve on the baseline performance by adapting the LM using manually transcribed anonymized voice queries. Our aim with these experiments is to perform both stylistic adaptation (from written to spoken query types) and also adjust the query distribution more towards the types of queries made on mobile devices, which also implicitly includes an element of temporal adpatation of the query distributions.

To facilitate LM adaptation we introduce a method using WFST operations to map from manually transcribed and segmented queries into a segmentation consistent with the current recognition system which then allows us to perform LM adaptation using data from both written and spoken sources.

2. Decoder improvements

In this section we describe our development of a WFST speech decoder that is optimized for the Japanese voice search task and Yahoo! Japan's server infrastructure. In common with other voice search endeavours[4] we chose to implement the WFST approach over lexical-tree decoding e. g.[9]. We show that the memory required to store a static WFST can quite easily be reduced through appropriate implementation decisions. Our target platform is a large number of 64-bit 48GB RAM servers which means that we can use relatively large WFSTs for decoding and also run multiple ASR processes on a single server.

2.1. Arc label compression

Our decoder is written in C++ and during development we discovered there were many C++-specific issues which had a significant effect on both speed and memory consumption, in particular with respect to the choice of STL containers and how memory is allocated and released.

Beyond these language-specifc details the most significant speed-up came from reducing the overall memory usage, which is dominated by having to store the static WFST in memory. As pointed out in [10] a simple storage structure typically uses 16 bytes per arc (4-byte integers for input label, output label and destination state and a 4-byte float for the weight) and 8 bytes per state (4-byte integers for an index to the arc information and for storing the number of arcs exiting a state). Instead of trying to compress the WFST storage globally we preserve the concept of an arc class and aim to minimize the memory required by an individual arc.

For all WFSTs we had encountered we observed that there were never more than 2^{24} unique combinations of input and output label values. Indeed, by far the most

frequent input/output pair has epsilons as labels. Epsilons on either input or output label also dominate the most frequent pairs. While the total possible combinations is bounded by the product of the vocabulary size and number of physical triphones most of these never occur in practice. Consequently, storing only the index lookup allows us to represent the input/output label combination using 3 bytes.

2.2. Arc weight compression

Moreover, in a range of experiments on the voice-search task we found that quantizing arc weights down to 6 bits produced no change in SER. For convenience, we chose to use 8 bits for quantization thus allowing us to pack into a 4-byte integer a 1-byte index (into the quantized 4-byte floating-point value) together with a 3-byte index to the input and output labels as shown in Fig. 1. Unpacking requires only trivial and fast bit-shifting and masking operations. The 4-byte type storing the destination state is left as-is for a total 8-byte structure, which is also appropriate for the 64-bit machines we are designing for. The WFST state structure is left unaltered to use 8 bytes

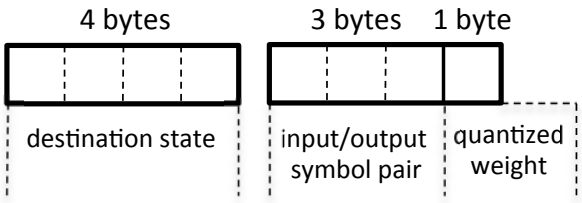

Figure 1: Compressed WFST arc class representation.

Alternative implementations such as those described in [10] can give greater reductions in memory use but at the expense of far greater complexity in terms of decoder implementation and WFST construction. We intentionally chose this compromise between performance and complexity.

Our static recognition network is a standard integrated phone-level WFST R constructed in the log semiring from C the context-dependency transducer, L the lexicon, and G the LM, and we use the silence transducer T given in Fig. 4 of [14] for silence modelling. Juicer[8] is used for building the component transducers and OpenFST[7] for the following WFST operations on them:

$$R = \pi(eps(min(det(C \circ det(L \circ min(det(G \circ T))))))) \quad (1)$$

where $\circ$ denotes composition, *det* determinization, *min* minimization, *eps* epsilon normalization, and π auxiliary symbol removal operations.

3. Baseline Experiments

In this section we describe the data used for training, development and testing our baseline speech recognition system and present a comparison of our internal decoder against two well-known open-source decoders.

3.1. Data

Morphological analysis of Japanese queries is used to segment text into word-like units comprising a surface form attached to one or more phonetic realisations of the surface form, for example:

東京都:トウキョウト and 一本:イッポン

where the surface realisation (orthography) of the word is to the left of the "**:**" and the phonetic realisation, represented using the *katakana* script, is shown to the right of "**:**". Combining surface and phonetic information in this way has been found to improve accuracy, particularly for recognizing digits and the names of people in context.

For the voice-search task we are fortunate to have access to a large amount of data. Yahoo! Japan is the largest search engine in Japan in terms of users and queries and the anonymized written query stream is available for training LMs. We use a training set (*train*) of 1.7B unique written queries segmented into units as described above. This translates into a total of 7.7B queries or 35B words (including sentence begin and end tokens).

In addition, since Mar. 2011, we have access to large numbers of spoken queries through the Yahoo! Japan iOS search application. 10,000 anonymized spoken utterances collected from this live system in Sept. 2011 are used for development (*dev*) and 10,000 utterances from Jan. 2012 are used for evaluation (*eval*). A further 1.6M spoken transcriptions collected between Mar. 2011 and Sept. 2011 are used for LM adaptation (*adapt*).

3.2. Vocabulary and Language model

We select the top 1M words (query-frequency weighted) from *train* to be the vocabulary. This gives an OOV-rate of around 2.5% on *dev*. We generate a 3-gram LM from *train* using an alternative implementation of entropy pruning[11], best described as entropy *growing*. This allows much larger LMs to be grown since all count statistics do not first need to be read into memory. On smaller LMs we have confirmed that growing does not result in any significant differences compared to pruning in terms of both the resulting number of N-grams, perplexity, and SER. Although our entropy growing approach is performed with respect to a Good-Turing smoothed Katz back-off model, we then re-smooth the grown model using absolute discounting. The pruned LM trained on *train* has 21.6M 2-grams and 13.2M 3-grams.

3.3. Acoustic model

Our acoustic model (AM) uses standard 3-state HMMs with 3000 decision-tree clustered context-dependent states with 32 Gaussians per state. Feature vectors are 38-dimension MFCCs (12 MFCC features, and the delta and delta-deltas of power and the MFCCs). Models are gender-independent and trained on 500 hours of anonymized voice queries collected between Mar. 2011 and Sept. 2011 using Maximum Likelihood training followed by Minimum Phone Error discriminative training[13].

3.4. Integrated WFST

Using the vocab, LM and AM described in Sections 3.2 and 3.3 we construct the integrated speech recognition WFST using the sequence of operations given in Eqn. 1. This results in a WFST with 137M arcs and 60M states. The number of unique input/output label pairs is 5.7M. All arc weights are then quantized into 8 bits using QccPack[1].

[1] `http://qccpack.sourceforge.net`

3.5. Japanese scoring metrics

Due to the segmentation and orthography issues associated with Japanese we typically use several metrics to determine performance. While others have used WebScore@N[12] as the test of recognition accuracy we choose not to use it since it is difficult to control for changes in the underlying search engine. Instead, we employ a form of sentence error rate (SER) which first tries to match the orthography character-for-character, then backs off to an exact pronunciation match if it fails. We also use the character error rate (CER).

3.6. Performance comparison

In Fig. 2 we plot SER vs. RTF on *eval* for our internal decoder (which always generates lattice traceback information) and two open-source decoders: (1) the WFST-based decoder Juicer v1.0.0 (decoderLite)[8] and (2) the lexical-tree based decoder Julius[9] v4.2.1 which uses a 2-gram forward pass followed by a 3-gram backward pass for decoding. For each decoder parameters (primarily beam, band, LM weight and insertion penalty (and also phone-end beam for Juicer)) were optimized on *dev* and results are reported on *eval*.

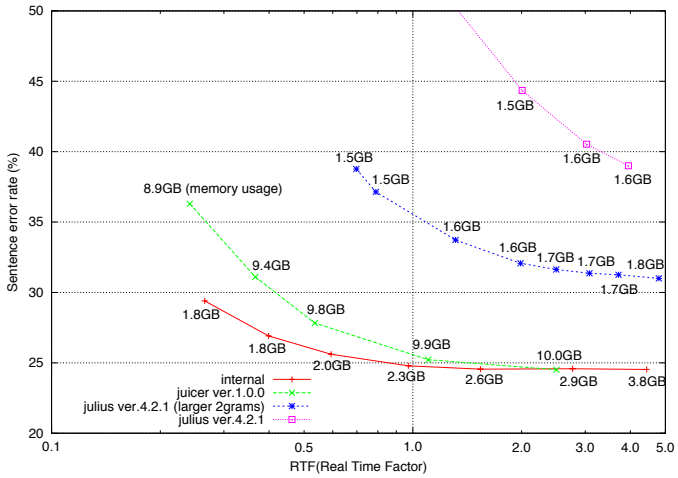

Figure 2: Sentence error rate vs. RTF curve of the three different recognizers (internal, Julius and Juicer) on *eval* over a range of decoder parameters optimized on *dev*.

All experiments were performed on a 2xCPU Intel(R) Xeon(R) X5675 @3.07GHz each with 6 cores, 12MB caches and 128GB RAM and running CentOS 5.4.3 64-bit. All decoders were compiled using GCC 4.1.2.

Comparing the performance at each decoder's real-time performance point we see that our internal decoder requires about 2.3GB memory whereas Juicer requires 9.9GB (330% more), and Julius around 1.6GB (30% less). To obtain better results with Julius (second curve from top) it was necessary to run the first pass with an unpruned forward 2-gram. Pruning the LMs to the same extent as used in the other decoders results in much worse performance (top-most curve). Memory usage and RTF go up with increasing beam for all decoders. In subsequent experiments we use the same configuration of our internal decoder running at 1xRT to generate lattices for subsequent rescoring.

4. LM Adaptation Experiments

In this section, we perform simple LM adaptation by interpolating the baseline LM which was trained only on written queries (*train*) with an LM trained on manually transcribed spoken utterances (*adapt*) as follows:

$$P(w \mid h) = \lambda \cdot P_{train}(w \mid h) + (1 - \lambda) \cdot P_{adapt}(w \mid h) \quad (2)$$

where λ is determined empirically for each size of adaptation data by minimizing SER or CER, accordingly, on *dev* lattices, generated using the baseline LM.

4.1. Japanese text segmentation / normalization

Typically, a Japanese morphological analyser is used to perform word segmentation of written text, but it has the following drawbacks for spoken text transcribed by humans. Firstly, it segments the orthographic representation of the input text and ignores any spoken information which is sometimes useful to select the correct word segmentation. For example, segmentation of the query 東京都 has two alternatives 東京:トウキョウ 都:ト and 東:ヒガシ 京都:キョウト (the pronunciation differs according to the segmentation: "tokyo to" vs. "higashi kyoto") but only the former is compatible with the original manual transcription. Secondly, the word definition in a morphological analyzer is optimized for its performance and efficiency. For example, the query 一本 is usually segmented into 一 and 本 by popular morphological analyzers. However, our ASR word definition favours 一本 as a single word since its pronunciation has a nasal sound change which we wish to preserve.

Suffice to say there is no agreed-upon definition of a word in Japanese. For manual transcriptions both the orthographic form and the spoken pronunciation are transcribed using whatever segmentation the transcriber deems appropriate. Manual transcription and segmentation will therefore likely differ both among transcribers and also automatic segmentation systems. Moreover, simply ignoring the initial manual segmentations by concatenating and re-segmenting automatically does not always work.

To integrate the manual transcriptions with our existing LMs, we first have to map to the required segmentation while preserving the human annotated surface form and pronunciation. To ensure consistent segmentation between human transcribed text and our existing integrated WFST we perform the following WFST operations:

- Step 1:
  - Prepare a WFST P from the pronunciation of transcribed query; input and output symbols are triphones.
  - Prepare a WFST O from orthography of transcribed query; input and output symbols are orthographic characters.
  - Prepare a dictionary WFST D; input symbols are orthographic characters and outputs are words.
- Step 2:
  - Compose P with the integrated phone-level recognition transducer R from Eqn. 1. Obtain an N-best candidate WFST N with triphone input labels and word output labels.
- Step 3:
  - Project output labels in N to give $proj(N)$ with word input and output labels.
- Step 4:

  - Compose D with $proj(N)$ and obtain an N-best candidate WFST $D \circ proj(N)$ with orthographic character input and word output labels.
- Step 5:
  - Compose O with $D \circ proj(N)$ and obtain an N-best candidate WFST $O \circ D \circ proj(N)$ compatible with both spoken and orthographic informations in the transcribed query.

If the result is not empty $O \circ D \circ proj(N)$ will have the same orthography and pronunciation as that given by the human transcriber and we can select the highest-scoring path which will have a segmentation consistent with the existing integrated recognition WFST.

4.2. Experiments

Ensuring that transcriptions from *dev* and *eval* are excluded, we sampled transcriptions from *adapt* in powers of 2, from 7,813 up to 1M transcriptions. For each sample size, we built a 3-gram LM using absolute discounting and Katz back-off and interpolated it with the baseline LM. Using interpolation weights in increments of 0.05 we re-scored *dev* set lattices to determine the interpolation weight which minimized SER or CER, as appropriate. We then re-scored *eval* set lattices using the same weight. The optimal interpolation weights, as determined on *dev*, the SER, CER and relative improvements over the baseline values on *eval* are shown in Table 1. We see there is a reasonably consistent reduction in both SER and CER with increasing number of adaptation transcriptions, with relative reductions of up to 4.6% in SER and 8.2% in CER when adapting with 1M spoken transcriptions.

# Trans.	Weight	SER/CER	Rel. Improv.(%)
Baseline	-	24.79/23.70	SER/CER
7,813	0.20/0.15	24.42/23.19	1.49/2.15
15,625	0.45/0.15	24.45/23.16	1.37/2.28
31,250	0.35/0.40	24.31/22.82	1.94/3.71
62,500	0.45/0.50	24.16/22.71	2.54/4.18
125,000	0.35/0.35	24.23/22.91	2.26/3.33
250,000	0.75/0.70	23.94/22.50	3.43/5.06
500,000	0.75/0.75	23.80/22.03	3.99/7.05
1,000,000	0.75/0.85	23.64/21.75	4.64/8.23

Table 1: SER and CER on *eval* set lattices re-scored using optimal interpolation weight determined on *dev* of each adaptation LM which was trained on different numbers of transcriptions.

5. Conclusion and Further Work

In this paper we have presented work on Japanese voice-search at Yahoo! Japan. We have described the improvements to our internal decoder in terms of reduced memory requirements for representing the static WFST by exploiting the distribution of input and output labels to compress the arc representation and quantizing arc weights down to 8 bits. Using identical LMs trained on 7.7 billion written Japanese search queries we compared the performance of our internal decoder against two well-known open-source decoders and showed that our decoder runs with a higher accuracy in real-time and uses 2.3Gb of memory; this is 76% less than the comparable open-source decoder.

To integrate spoken transcriptions with our existing LM we presented a sequence of WFST operations to map the transcriptions into a consistent segmentation. Using this technique we then performed simple LM adaptation of the baseline LM using up to 1 million transcribed voice queries which gave up to 4.6% relative reduction in sentence error rate and 8.2% relative reduction in character error rate.

In future work we will look at how to perform more effective language model adaptation and also use automatic, instead of manually, transcribed voice queries.

6. References

[1] Schuster, M., "Speech Recognition for Mobile Devices at Google", In PRICAI 2010, LNAI 6230 2010.

[2] Chelba, C. and Schalkwyk, J. and Harb, B. and Parada, C. and Allauzen, C. and Riley, M. and Xu, P. and Brants, T. and Ha, V. and Neveitt, W., "Language Modeling for Automatic Speech Recognition Meets the Web: Google Search by Voice", 2011

[3] Mamou, J. and Sethy, A. and Ramabhadran, B. and Hoory, R. and Vozila, P., "Improved Spoken Query Transcription using Co-Occurrence Information", In Proceedings of Interspeech 2011.

[4] Chelba, C. and Schalkwyk, J. and Brants, T. and Ha, V. and Harb, B. and Neveitt, W. and Parada, C. and Xu, P., "Query Language Modeling for Voice Search", In Proceedings of the 2010 IEEE Workshop on Spoken Language Technology.

[5] Schuster, M., "Japanese and Korean Voice Search", In Proceedings of ICASSP 2012.

[6] Acero, A. and Bernstein, N. and Chambers, R. and Ju, Y. and Li, X. and Odell, J. and Nguyen, P. and Scholz, O. and Zweig, G., "Live Search for Mobile: Web Services by Voice on the Cellphone", In Proceedings of ICASSP 2007.

[7] Allauzen, C. and Riley, M. and Schalkwyk, J. and Skut, W. and Mohri, M., "OpenFST: A General and Efficient Weighted Finite-State Transducer Library", In Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007), volume 4783 of Lecture Notes in Computer Science, pages 11-23. Springer, 2007. http://www.openfst.org.

[8] Moore, D. and Dines, J. and Magimai Doss, M. and Vepa, J. and Cheng, O. and Hain, T., "Juicer: A Weighted Finite-State Transducer speech decoder", In Proceedings of 3rd Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms MLMI'06, 2006.

[9] Lee, A. and Kawahara, T., "Recent Development of Open-Source Speech Recognition Engine Julius", In Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2009.

[10] Caseiro, D., "WFST Compression for Automatic Speech Recognition", In Proceedings of Interspeech 2010.

[11] Stolcke, A., "Entropy-based Pruning of Backoff Language Models", In Proceedings of the 1998 DARPA Broadcast News Transcription and Understanding Workshop.

[12] Schalkwyk, J. and Beeferman, D. and Beaufays, F. and Byrne, B. and Chelba, C. and Cohen, M. and Garrett, M. and Strope, B., "Google Search by Voice: A Case Study", In Advances in Speech Recognition: Mobile Environments, Call Centers, and Clinics, Amy Neustein, Ed. Springer-Verlag, 2010.

[13] Povey, D. and Woodland, P.C., "Minimum phone error and I-smoothing for improved discriminative training", In Proceedings of ICASSP 2002.

[14] Allauzen, C. and Mohri, M. and Riley, M. and Roark, B., "A generalized construction of integrated speech recognition transducers", in Proceedings of ICASSP 2004.