Lattice Rescoring for Speech Recognition Using Large Scale Distributed Language Models

Euisok Chung Hyung-Bae Jeon Jeon-Gue Park and Yun-Keun Lee
Speech Processing Research Team, ETRI, 138 Gajeongno, Daejeon, 305-700, KOREA
eschung@etri.re.kr, hbjeon@etri.re.kr, jgp@etri.re.kr and yklee@etri.re.kr

ABSTRACT

In this paper, we suggest a lattice rescoring architecture that has features of a Trie DB based language model (LM) server and a naïve parameter estimation (NPE) to integrate distributed language models. The Trie DB LM server supports an efficient computation of LM score to re-rank the n-best sentences extracted from the lattice. In the case of NPE, it has a role of an integration of heterogeneous LM resources. Our approach distributes LM computations not only to distribute LM resources. This is simple and easy to implement and maintain the distributed lattice rescoring architecture. The experimental results show that the performance of the lattice rescoring has improved with the NPE algorithm that can find the optimal weights of the LM interpolation. In addition, we show that it is available to integrate n-gram LM and DIMI LM.

KEYWORDS : lattice rescoring, distributed language model, large scale language model

Proceedings of COLING 2012: Posters, pages 217–224,
COLING 2012, Mumbai, December 2012.

1 Introduction

The speech dictation with over multi-million words requires the large-scale language model. This need has a few problems such as a high computation time and a memory limitation. Automatic speech recognition for the multiple simultaneous accesses occupies the memory as multi-processes and uses multi-core capability of the CPU to guarantee high performance service. Hence, the limitation of the system resource requires the distributed approach for the large-scale language model. Previous researches have shown that the distributed modeling approach is available to avoid these problems.

In the case of language model researches, the distribution approach focuses on the client/server paradigm with splitting a training corpus as a technique of suffix array (Zhang, 2006 and Emami, 2007). These approaches depend on the distributed n-gram count servers; on the other hand, there is a more sophisticated technique to alleviate the burden of network communication. It uses MapReduce programming model to save and serve the smoothed probability of n-gram (Brants, 2007). These researches have presented the distributed architecture for the n-gram based language model. In the case of composite language model, there is a research, which simultaneously accounts for lexical information, syntactic structure and semantic content under a directed Markov random field paradigm (Tan, 2011). In addition, the composite language model approach showed the limitation in the training time, which takes 25.6 hours for the EM algorithm to build model of 230M corpus in the cloud environment.

In this paper, we suggest a lattice rescoring architecture that has features of a Trie DB based language model (LM) server and naïve parameter estimation (NPE) to integrate distributed language models. We use this architecture for speech recognition. Therefore, the multi-stage lattice rescoring approach is prerequisite. The Trie DB language model server has a role of efficient computation of LM score to re-rank the n-best sentences extracted from the lattice. In the case of NPE, it has a role of an integration of heterogeneous LM resources.

2 Lattice rescoring architecture

2.1 Lattice rescoring flow

The process of lattice rescoring begins with the automatic speech recognition (ASR) that recognizes the input speech and generates the lattice that is a weighted directed acyclic graph where represents the ASR results. With the lattice input, the am/lm decoupling step splits acoustic model (AM) and language model (LM) scores of the lattice for the lattice rescoring since in the LM rescoring stage, we only use AM scores of the input lattice. After that, it extracts the N-best list from the lattice. The rescoring step rescores the sentence scores of the N-best list with large scaled LM resources. Finally, it reorders the n-best list according to the new scores.

The rescoring step uses the AM scores of n-best sentences and new LM scores computed in distributed LM servers. The LM server and rescoring module communicates through stream sockets. The LM servers return each LM scores when it receives n-best sentences. The rescoring module re-ranks the n-best sentences after interpolating new LM scores received from the distributed LM servers.

The rescoring flow depends on two approaches, one is the LM interpolation parameter estimation and the other is the LM Trie DB. The step of the LM interpolation parameter estimation computes the interpolation weights in the back-end step with the correct ASR result scripts. We propose Naïve parameter estimation algorithm to estimate the LM interpolation weights. In the case of LM Trie DB, we build LM as a Trie DB that guarantees high performance and light footprint. Figure 1 describes the flow of lattice rescoring.

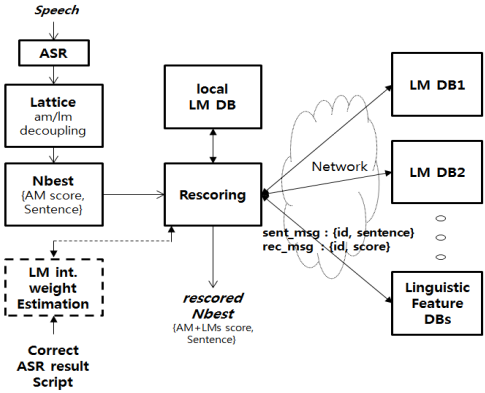

FIGURE 1 – System architecture for a lattice rescoring

2.2 Lattice Generation and AM/LM Decoupling

We implement the unit of generating lattices considering high performance. The lattice is built at the ASR decoding step without the increase of memory and computation. The decoder generates the lattice using the recognized word path at the backtracking step. It attaches completed word paths to the 1-best recognized word at the specific time according to the accumulated likelihood score.

The lattice link has the likelihood score that is a summation of AM and LM scores with the proportion determined empirically. We decouple the likelihood score with the original LM of the ASR decoder since we cannot improve the result of lattice rescoring when we maintain the original LM scores. Therefore, the basic step of the lattice rescoring is the replacement of the lattice LM score with other LM resources.

2.3 LM Trie DB Server

We propose LM Trie DB server. It consists of two components; one is the LM Trie DB and the other is the service function. The LM Trie DB is built by converting the ARPA format for language model representation into a Trie structure. In the case of the server function, it computes the LM scores for the n-best sentences in a style of dynamic programing. In runtime, the DB is loaded in the memory space to deal with the requests of LM value computation for the N-best sentence list. As a DB structure, we use a double-array Trie approach (Aoe, 1992).

The basic schema of LM Trie DB is a pair of key string and data string. The 1-gram entry has "word" as a key and "prob_backoff_winx" as a data; "word" is a unigram word string, "prob" is a

LM probability, "backoff" is a value of backoff and "winx" is the index of this entry which is used in n-gram entries. In the case of 2-gram entry, the key string is "winx_winx". It means that the key string is composed of two 1-gram word indexes. Also, it has "winx2" for a 2-gram index used in 3-gram entries. Table 1 shows the schema of LM Trie DB.

Key	Data
word	prob backoff winx
winx winx	prob backoff winx2
winx2 winx	prob backoff winx3
winx3 winx	prob backoff winx4
winx4 winx	prob backoff winx5
winx5 winx	prob backoff

TABLE 1 – Schema of LM Trie DB

The dynamic chart for the computation of LM score is described in Figure 2. This figure shows to compute LM score for the input string with 4-gram LM. First line shows input string. The LM values are presented from 2nd layer to 4th layer. The cell filled with backoff b_i and probability p_i. The arrow shows the computation with previous layer scores when there is no n-gram entry in LM DB. We denote a probability of a dynamic chart as a DC(n-gram, p_n) and a backoff value as a DC(n-gram, b_n).

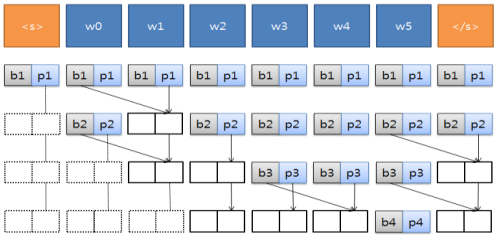

FIGURE 2 – Dynamic chart for the computation of LM score

When the LM Trie DB server receives the request of the computation of LM value, first, it searches 1-gram data in the LM Trie DB with input sentence <s> w_0 w_1 w_2 .. w_n </s>. Then, it searches 2-gram data. When it cannot find the 2-gram data, it fills the slot of the dynamic chart with the backoff and probability of each composed 1-gram; DC(w_0w_1, p_2) ← DC(w_1, p_1) + DC(w_0, b_1). If there is no backoff value, the previous probability transfers to the current slot; DC($w_0w_1w_2$, p_3) ← DC(w_1w_2, p_2). Finally, the summation of last slots is the LM value of the input sentence; ∑ DC(-, p_4). This procedure is same with a normal procedure for backoff in LM. The difference is that the DC computation depends on the schema of LM Trie DB. The higher n-gram DB search uses the "winx" of the lower n-grams.

2.4 Distributed LM interpolation

We propose naïve parameter estimation (NPE) algorithm for the integration of distributed LMs. The goal of NPE estimates the optimal interpolation weights of the distributed LMs to the evaluation set. Simply, NPE uses the accuracy of the ASR to the evaluation set with each LM. The idea is that the update of the weight of the LM is multiplicative (high change) when the accuracy of ASR decreases and the other is additive (small change) when the accuracy of ASR

increases. Although we adopt simple approach to estimate LM interpolation weights, it can find the optimal weights of all LMs in a few iterations.

In addition, we can process the NPE in distributed environment since the ASR evaluation function only uses the network procedures. The evaluation function, do_eval(), sends the message of LM score computation and receives the message of LM score from each LM server. Although the NPE sends the network calls iteratively to the LM servers, it can efficiently process the task since the NPE uses not only the distributed LM resources but also the distributed LM computation.

```
 0: E := {e1..en}
 1: W ← initialize()   # W := {w1..wn}
 2: ΔW ← W * c         # 0 < c < 1
 3: acc_old ← do_eval(W, E)
 4: for itr = 0 to max_iteration do
 5:   W' ← W
 6:   for i = 0 to number of LMs do
 7:      w'i ← wi + Δwi
 8:      acc_new ← do_eval(W', E)
 9:      if (acc_old-acc_new > 0) then
10:         Δwi ← -Δwi * random()
11:      else
12:         Δwi ← Δwi + random()
13:      end if
14:   end for
15:   W ← W + ΔW
16:   acc_old ← do_eval(W, E)
17:   if (acc_old is max) then
18:      Wmax ← W
19:   end if
20: end for
21: return Wmax
```

FIGURE 3 – Naïve parameter estimation algorithm.

The NPE algorithm is described in Figure 3. Firstly, it initializes the interpolation weights W as many as the number of LMs (line1). In addition, it initializes ΔW which compute by multiplying constant value c ($0 < c < 1$) to the interpolation weight W (line2). The last stage of the initialization is to get the first accuracy with the initialized W (line3). At this time, do_eval() evaluates the evaluation set E that is the set of the lattice and correct script pairs.

The evaluation step, do_eval(), extracts n-best from the input lattice and then rescores the n-best with the distributed LMs. It sends the n-best sentences to the distributed LM servers and receives each LM scores computed by the LM servers. Then, it computes the score of LM interpolation with W to re-rank the n-bests. Finally, it can compare the correct scripts to find the accuracy.

The weight estimation step processes iteratively. It try to evaluate with the updated weight in each LMs (line7 ~ line8). If new updated weight cannot show the better accuracy, it processes a multiplicative decrease of the weight (line10). On the other hand, if new weight shows the better result, it processes an additive increase of the weight (line12). We use the random scale to change the weight value in order to avoid the stalled state, which is a repetition of two weight values.

After deciding all LM weights, NPE gets new evaluation value (line15 ~ line16). Then, if the value is the maximum, it saves the value as W_{max}(line18). Let the rescoring function for a sentence s be res(s). We define:

$$res(S) = am(S) + \sum_{i=1}^{n} w_{npe}(i) \cdot lm_i(S)$$

Where, $w_{npe}(i)$ is the NPE determined i^{th} weight of the distributed LM and $lm_i(s)$ is the result of i^{th} distributed LM to the sentence s. am(s) is the decoupled AM score to the sentence s. These values are in the log domain so that we can add them as shown in the equation.

3 Evaluation

3.1 Evaluation Set and LMs

The evaluation set also consists of four domains such as email, news, Q&A and twitter. It has 2,000 clean Korean speeches independent of LM training corpus. We converts the speeches into HTK standard lattice format (SLF) files with wFST-based Korean speech recognizer, which uses small LM. We prepare two evaluation sets as described in Table 2. EVAL1 uses all sentences and EVAL2 divides the evaluation set into a train/development set and a test set.

# of speech		email	news	Q&A	twitter	All
EVAL1		200	400	400	1000	2000
EVAL2	train	150	300	300	750	1500
	Test	50	100	100	250	500

TABLE 2 – Preparation of two evaluation sets.

We select a vocabulary set for building language models. The vocabulary has 1.3 million entries which extracts from the corpus of 3.3 billion words with a coverage of 99.84% of the corpus. We use the 3.3 billion words corpus as a training set for language models in this evaluation. The domain of the corpus consists of twitter, news, community and Q&A. The training corpus is built by crawling from the web sites

We builds two n-gram language models for the evaluation; one is Small LM (1.3m 1gram, 4.5m 2gram, 2.3 3gram), and the other is Big LM (1.3m 1gram, 42.1m 2gram, 45.8m 3gram). In addition, we build the distance independent mutual information LM (DIMI LM) (GuoDong, 2004), which has 121 million pairs extracted from the training data within the 6 words distance.

3.2 Lattice Rescoring

We use EVAL1 to evaluate our distributed LM architecture. In this experiment, EVAL1 is a training set of the NPE algorithm to estimate interpolation weights for the LMs. Also, EVAL1 is a test set of this experiment. Table 3 shows the result of the lattice rescoring tests.

type		email	news	Q&A	twitter	All
1	AM	85.13	80.34	83.59	85.6	84.32
2	AM+Small LM	86.93	83.17	86.49	87.16	86.35
3	AM+Big LM	88.3	84.67	87.4	88.15	87.41
4	AM+Big LM+DIMI LM	88.41	85.81	87.59	88.28	87.73
5	Big LM+DIMI LM (no AM)	85.1	83.75	85.7	85.28	85.13

TABLE 3 – Evaluation with EVAL1 (accuracy %, top1)

The result of type1 is the ASR accuracy with only AM scores. The result of type2 is the baseline accuracy since it is the performance of ASR with small LM. We use the NPE algorithm from type3 to type5. The gain of the accuracy is 1.06% when we apply the Big LM to replace small LM in type3. The test type4, the accuracy of the all test sentences increases in small. However, in news domain, the gain of the accuracy is 1.14%. The result of type5 shows the importance of AM scores. The test cannot improve the result of lattice rescoring when we ignore AM scores in the input lattice.

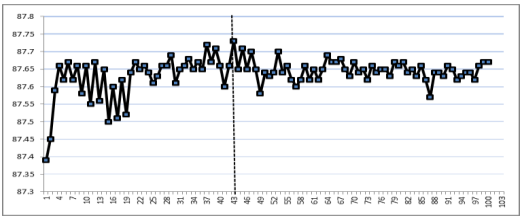

FIGURE 4 – Naïve parameter estimation algorithm.

Figure 4 shows the oscillation of the accuracy when NPE estimates the interpolation weights to the big LM and DIMI LM (type4). The NPE finds the optimal weights in the 43th iteration and the duration time is not over 20 minutes. Although the NPE cannot maintain the optimal accuracy, the result shows that it is available to find the appropriate interpolation weights in the short-term. This evaluation shows that the NPE can integrate a long-distance LM that is different with n-gram based LMs. In addition, the algorithm can estimate the interpolation weights to the multiple LM resources.

type			email	news	Q&A	twitter	All
Baseline	AM+Small LM	Train	86.15	83.71	86.26	87.23	86.29
		Test	87.84	82.01	85.57	86.48	85.97
Lattice Rescoring	AM+Big LM +DIMI LM	Train	87.65	86.28	87.83	88.23	87.63
		Test	88.87	83.61	86.21	88.18	87.37

TABLE 4 – Evaluation with EVAL2 (accuracy %, top1), NPE with Train Set

The evaluation with EVAL2 described in Table 4. In this experiment, we apply NPE only to the train set. The result shows that the gain of accuracy of test set is 1.4% when the gain of accuracy of train set is 1.34%. From this test, we find that the NPE cannot guarantee the optimal weight of test set with only train set because of the over-fitting problem; the accuracy of all test is 87.47 when we apply NPE to test set. However, the result shows consistency in the gain of accuracy in all domains.

	Distributed LM		Non-distributed LM	
	Acc. %	Time(sec)	Acc. %	Time(sec)
1st	88.44	180	88.44	206
2nd	88.33	164	88.47	249
3rd	88.36	184	88.36	193
Avg.	88.37	176	88.42	216

TABLE 5 – Distributed LM vs. Non-distributed LM

In addition, we test the email set of EVAL1 considering the comparison of distributed LM and non-distributed LM. We test it 3 times of NPE with 30 iterations. The total number of LM score computation is 394,581 in one NPE process. The type of experiment is same with type 4 in Table 4. From the Table 5, we find that the reduction of the time is 18%. In the case of accuracy, there is only marginal difference between two tests.

If the Non-distributed LM is 1st-pass big LM based ASR, then the result of this test, EVAL1 email set, is 89.16% accuracy; EVAL1 all set is 88.20%. The two-pass approach such as the lattice rescoring cannot overcome 1st pass approach of the big LM since the small LM based ASR cannot show the coverage of n-gram path of big LM. However, in this paper, our assumption is the case that it is not possible to use the approach of 1st pass big LM ASR.

The main feature of our approach is to distribute LM computations not only to distribute LM resources. The rescoring client sends the k numbers of n-best sentences to the k numbers of LM servers. The LM servers return LM scores of the n-best sentences to the client. The computation of a LM scoring is only occurred in the servers in parallel with each other. This is simple and easy to implement and maintain the distributed lattice rescoring architecture.

Conclusion and perspectives

In this paper, we proposed the lattice rescoring architecture for applying the large scale distributed language model to the speech recognition. AM/LM decoupling approach of a lattice is required to replace large scale LMs with first-pass small LM. In the distributed LM server, we adopted socket-streaming approach and the Trie-based memory DB for LM. Finally, we suggested the naïve parameter estimation algorithm for the interpolation of multiple LMs. The evaluation showed the appropriate gain using NPE algorithm that can find the optimal weights of the LM interpolation. Also, we showed the integration between n-gram LM and DIMI LM. In the future, we will improve the NPE algorithm in various domains. Domain adaptation technique can be one of them.

References

Aoe, J., Morimoto, K. and Sato, T. (1992). An efficient implementation of trie structures, Software Practice & Experiments, 22(9): 695-721.

Emami, A., Papineni, K. and Sorensen, J. (2007). Large-scale distributed language modeling. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing 2007.

Guodong, Z. (2004). Modeling of Long Distance Context Dependency. In Proceedings of the *20th international conference on computational linguistics, COLING '04*, page 92, 2004.

Tan, M., Zhou, W., Zheng, L. and Wang, S. (2011). A Large Scale Distributed Syntactic, Semantic and Lexical Language Model for Machine Translation. In 49th Annual Meeting of the Association for Computational Linguistics(ACL), 201-210.

Zhang, Y., Hildebrand, A. S. and Vogel, S. (2006). Distributed language modeling for Nbest list re-ranking. In Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, pages 216–223.