REAL-TIME ONE-PASS DECODING WITH RECURRENT NEURAL NETWORK LANGUAGE MODEL FOR SPEECH RECOGNITION

Takaaki Hori, Yotaro Kubo, and Atsushi Nakamura

NTT Communication Science Laboratories, NTT Corporation
2-4, Hikaridai, Seika-cho, Soraku-gun, Kyoto, Japan
{hori.t, yotaro.kubo, nakamura.atsushi}@lab.ntt.co.jp

ABSTRACT

This paper proposes an efficient one-pass decoding method for real-time speech recognition employing a recurrent neural network language model (RNNLM). An RNNLM is an effective language model that yields a large gain in recognition accuracy when it is combined with a standard n-gram model. However, since every word probability distribution based on an RNNLM is dependent on the entire history from the beginning of the speech, the search space in Viterbi decoding grows exponentially with the length of the recognition hypotheses and makes computation prohibitively expensive. Therefore, an RNNLM is usually used by N-best rescoring or by approximating it to a back-off n-gram model. In this paper, we present another approach that enables one-pass Viterbi decoding with an RNNLM without approximation, where the RNNLM is represented as a prefix tree of possible word sequences, and only the part needed for decoding is generated on-the-fly and used to rescore each hypothesis using an on-the-fly composition technique we previously proposed. Experimental results on the MIT lecture transcription task show that our proposed method enables one-pass decoding with small overhead for the RNNLM and achieves a slightly higher accuracy than 1000-best rescoring. Furthermore, it reduces the latency from the end of each utterance in two-pass decoding by a factor of 10.

***Index Terms*—** Speech recognition, Recurrent neural network language model, Weighted finite-state transducer, On-the-fly rescoring

1. INTRODUCTION

Language models are indispensable for large-vocabulary continuous-speech recognition. Such models, which are usually based on n-gram statistics, provide prior probabilities of hypothesized sentences to disambiguate their acoustical similarities. To build an n-gram model, text corpora are used to estimate the probability of a word's occurrence conditonal on the preceding n-1 words, where n is typically 3 or 4.

On the other hand, continuous space language models based on neural networks have attracted increased attention in recent years [1, 2, 3, 4, 5]. With this approach, word indices are mapped to a continuous space and word probability distributions are estimated as smooth functions in that space. Consequently, the approach makes it possible to provide better generalization for unseen n-grams [1]. A recurrent neural network language model (RNNLM) is a promising instance of such continuous space language models. An RNNLM has a hidden layer with re-entrant connections to itself with one word delay. Hence, the activations of the hidden units play a role of *memory* keeping a history from the beginning of the speech. Accordingly, the RNNLM can robustly estimate word probability distributions by taking long-distance interword dependencies into account. Mikolov, et al. reported that RNNLMs yielded a large gain in recognition accuracy when combining it with a standard n-gram model [2].

However, an RNNLM needs many more computations in the decoding phase than standard n-gram models. First, as a common problem when using continuous space language models, the computational cost to obtain a word probability is much more expensive than the table look-up in the case of n-gram model. This is because we need to compute activations of neurons and feed them to the higher layers after computing the sigmoid or the softmax activation function for each neuron. Second, especially with an RNNLM, the search space in Viterbi decoding grows exponentially with the length of the recognition hypotheses and makes computations prohibitively expensive because every word probability distribution based on an RNNLM is dependent on the entire history from the beginning of the speech.

Therefore, speech recognition with RNNLMs is usually performed by a two-pass search strategy based on N-best rescoring [2, 6] or lattice rescoring [7]. But the two-pass strategy may not be suitable for online applications, because its 2nd-pass search requires a certain computation that delays the system responses. If we can take a one-pass search strategy, the latency of the system is potentially reduced dramatically. In addition, the recognition results can be decided earlier without waiting for the end of each utterance by using a technique in [8]. This option is advantageous in real-time systems that display the recognition result as soon as possible on a word-by-word basis [9].

One possible way to perform one-pass decoding is to approximate the RNNLM to a back-off n-gram model in advance, and convert it to a weighted finite-state transducer (WFST) [10]. However, since this approach quantizes different histories in the continuous space by K-means clustering and entropy-based pruning, it sacrifices some recognition accuracy due to the quantization error. Furthermore, since the conversion step requires a long time [1], it may be difficult to use dynamic RNNLMs [4] based on unsupervised adaptation in on-line applications.

In this paper, we present another approach that enables one-pass decoding with an RNNLM without approximation. The approach is to represent the RNNLM as a prefix tree of possible word sequences in WFST form, and generate only state transitions needed during decoding, where the transition weight is directly computed with the RNNLM. The transitions are used to rescore each partial hypothesis using an on-the-fly composition technique that we previously proposed [11, 12, 13]. With this method, the number of hypotheses in Viterbi decoding is basically independent of the entire histories kept for the RNNLM unlike the case of standard on-the-fly composition [14, 15]. Accordingly, efficient one-pass decoding is possible with small overhead for the RNNLM. Furthermore, since we do not have

[1] In [10], it took few minutes and a few hours depending on the vocabulary size, the number of centroids, and the pruning threshold.

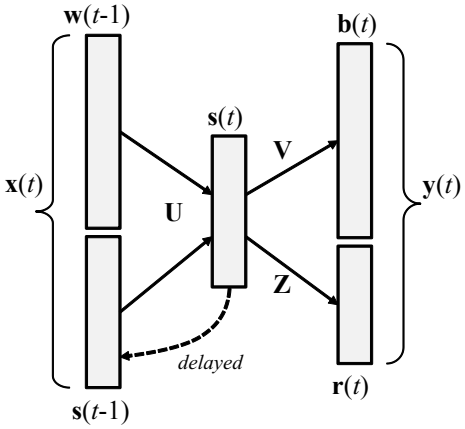

Fig. 1. Class-based RNNLM

to convert the RNNLM to a static WFST beforehand, we can start decoding immediately even if the RNNLM is dynamically updated. To our knowldege, there is no other one-pass decoder that directly uses an RNNLM without approximation.

This paper is organized as follows. Section 2 describes the basic structure of an RNNLM. Section 3 presents a WFST representation of an RNNLM and a method to generate its transitions on-the-fly from it. Section 4 reviews the one-pass decoding strategy based on the efficient on-the-fly composition technique we proposed before and mentions the case generating state transitions from an RNNLM. Section 5 shows our experimental results on a lecture transcription task, and Section 6 concludes this paper.

2. RECURRENT NEURAL NETWORK LANGUAGE MODELS

In this work, we adopted the class-based RNNLM architecture [3] depicted in Fig. 1. The input vector for time index t is represented as

$$\mathbf{x}(t) = [\mathbf{w}(t-1)^T, \mathbf{s}(t-1)^T]^T, \tag{1}$$

where $\mathbf{w}(t-1)$ denotes the last word in the 1-of-N coding and $\mathbf{s}(t-1)$ denotes the previous hidden layer activation vector.

The current hidden layer activation vector $\mathbf{s}(t)$ can be computed as

$$\mathbf{s}(t) = \mathbf{f}\left(\mathbf{U} \cdot \mathbf{x}(t)\right), \tag{2}$$

where $\mathbf{U}$ is the weight matrix between the input and hidden layers and $\mathbf{f}(\cdot)$ denotes a sigmoid function that computes the sigmoid for each element in a given vector.

Output vector $\mathbf{y}(t)$ consists of word and class probability vectors:

$$\mathbf{y}(t) = [\mathbf{b}(t)^T, \mathbf{r}(t)^T]^T, \tag{3}$$

which can be obtained as

$$\mathbf{b}_k(t) = \mathbf{g}\left(\mathbf{V}_k \cdot \mathbf{s}(t)\right) \tag{4}$$

$$\mathbf{r}(t) = \mathbf{g}\left(\mathbf{Z} \cdot \mathbf{s}(t)\right), \tag{5}$$

where $\mathbf{b}_k(t)$ is the sub-vector of word probability vector $\mathbf{b}(t)$, $\mathbf{V}_k$ is the sub-matrix of weight matrix $\mathbf{V}$ to the word output layer, and k denotes the class index. $\mathbf{r}(t)$ is the class probability vector, and $\mathbf{Z}$ is the weight matrix to the class output layer. $\mathbf{g}(\cdot)$ denotes a softmax function that computes the softmax over the elements in a given vector.

Algorithm 1 RnnLMArcs(p, w)

if $\delta(p, w)$ *is not defined* **then**
  $q \leftarrow |Q|$
  $Q \leftarrow Q \cup \{q\}$
  $\delta(p, w) \leftarrow q$
else
  $q \leftarrow \delta(p, w)$
end if
if $\mathbf{s}(p)$ *is not computed* **then**
  $p' \leftarrow$ PARENTOF(p)
  $\mathbf{s}(p) \leftarrow \mathbf{f}(\mathbf{U} \cdot [\mathbf{w}(p')^T, \mathbf{s}(p')^T]^T)$
  $\mathbf{r}(p) \leftarrow \mathbf{g}(\mathbf{Z} \cdot \mathbf{s}(p))$
end if
if $\mathbf{b}_{c(w)}(p)$ *is not computed* **then**
  $\mathbf{b}_{c(w)}(p) \leftarrow \mathbf{g}(\mathbf{V}_{c(w)} \cdot \mathbf{s}(p))$
end if
$P_{\mathrm{rnn}}(w|p) \leftarrow b_{c(w)i(w)}(p)\, r_{c(w)}(p)$
if $w =$ </s> **then**
  $F \leftarrow F \cup \{q\}$
end if
return $\{(p, w, w, -\log P_{\mathrm{rnn}}(w|p), q)\}$

Finally, the following is the word occurrence probability of $w_{k\ell}$, which is the ℓ-th word in class c_k:

$$P(w_{k\ell}|\mathbf{w}(1), \ldots, \mathbf{w}(t-1)) \equiv P(w_{k\ell}|c_k, \mathbf{s}(t))P(c_k|\mathbf{s}(t)) = b_{k\ell}(t)r_k(t), \tag{6}$$

where $b_{k\ell}(t)$ and $r_k(t)$ are elements of $\mathbf{b}_k(t)$ and $\mathbf{r}(t)$, respectively.

With this class-based architecture, the computation for propagating activations from the hidden layer to the output layer can be reduced since we need to handle only the words in the class of the current word to compute the softmax function rather than all the words in the vocabulary.

3. WFST GENERATION FROM RNNLM

For using an RNNLM in one-pass decoding, we utilize its straightforward representation in WFST form, i.e., a prefix tree of possible word sequences in the vocabulary. However, the size of the WFST will grow exponentially with the length of the sequences. Suppose the vocabulary size is 50,000 and the length of the sequences is 10. The number of state transitions in the WFST will become $50,000^{10}$. Since it is not feasible to generate all such transitions, we generate only part of them as necessary during decoding.

We present an algorithm for on-demand arc generation from an RNNLM. To explain it, we first define the terminology related to WFST [16], which is a finite state network that associates input and output labels on each arc weighted with a minus log probability value. A WFST is defined over semiring $\mathbb{K}$ by an 8-tuple $(\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$, where Σ is a finite set of input labels, Δ is a finite set of output labels, Q is a finite set of states, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states, $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ is a finite multi-set of transitions, $\lambda : I \rightarrow \mathbb{K}$ is an initial weight function, and $\rho : F \rightarrow \mathbb{K}$ is a final weight function. "ε" is a meta symbol that indicates there is no symbol to input or output.

We assume here that an initial WFST only exists with initial state 0, i.e., $Q = I = \{0\}$. For any state p and word w, if a preceding state of p is already made, a WFST transition that accepts w from state p can be obtained with function RnnLMArcs(p, w).

In lines 1-7, destination state q is prepared, where $\delta(p, w)$ is a function that returns an individual state number depending on the p and w pair. In lines 8-12, activations are computed for the hidden layer and the class output layer for state p if they have not been computed yet. Compared to the equations in Section 2, time index t is replaced with state index p. PARENTOF$(\cdot)$ at line 9 returns the source state of p, where p' corresponds to $t-1$ in Section 2. In lines 13-15, the activations for the word output layer are computed over the words in $c(w)$, which is the class that w belongs to. On line 16, word probability $P_{rnn}(w|p)$ is computed based on Eq. (6), where $i(w)$ indicates that w is the $i(w)$-th word in the class. In lines 17-19, state q is set as a final state if w is a symbol representing a sentence end. The resulting transition is returned at line 20.

If we use RNNLM-based probabilities in combination with standard n-gram probabilities, we can just insert the following statement after line 16,

$$P_{\rm comb}(w|p) = \lambda P_{\rm rnn}(w|p) + (1-\lambda) P_{\rm ngram}(w|\phi(p)), \qquad (7)$$

and replace $P_{\rm rnn}$ with $P_{\rm comb}$ in line 20, where λ is a scaling factor to balance the RNNLM and n-gram probabilities and $\phi(p)$ represents $n-1$ word history of state p to calculate n-gram probability $P_{\rm ngram}(w|\phi(p))$.

4. SEARCH STRATEGY FOR ONE-PASS DECODING

Next we introduce an efficient on-the-fly composition technique, called *on-the-fly hypothesis rescoring* [13], to employ an RNNLM with small overhead in one-pass decoding. Given a transition e, we denote its input label by $i[e]$, its output label by $o[e]$, its origin state by $p[e]$, its destination state by $n[e]$, and its weight by $w[e]$. A hypothesis during the search $h = e_1, \ldots, e_k$ is a path along consecutive transitions from the initial state: $p[e_1] = i$, $n[e_{j-1}] = p[e_j]$, $j = 2, \ldots, k$. We extend $n[\cdot]$ and $p[\cdot]$ to paths as $n[h] = n[e_k]$ and $p[h] = p[e_1]$. We also extend $o[\cdot]$ and $w[\cdot]$ to paths as $o[h] = o[e_1] \cdot \cdots \cdot o[e_k]$ and $w[h] = w[e_1] + \cdots + w[e_k]$. We sometimes use a subscript indicating the corresponding WFST for $i[\cdot]$, $o[\cdot]$, $p[\cdot]$, $n[\cdot]$ and $w[\cdot]$ for disambiguation.

Suppose transducers A and B can be composed. In our approach, the Viterbi search is performed for A but not for composite transducer $A \circ B$ as in standard approaches. Let h be a partial hypothesis produced by A, which has accumulated weight $w_A[h]$. In the search process, h is linked to a list of *co-hypotheses* that are generated from B by taking symbol sequence $o_A[h]$ as B's input. The decoder rescores h using the minimally weighted co-hypothesis in the list. The rescoring is performed efficiently by managing each hypothesis h and co-hypothesis list $g[h]$.

During decoding, the following basic procedure is used to generate each hypothesis and rescore it by using its related co-hypotheses. When a new hypothesis h' is generated by adding a transition e originating from $n_A[h]$, the weight of h' can be calculated as $w_A[h'] = w_A[h] + w_A[e]$. If the transition e outputs nothing ($o_A[e] = \varepsilon$ and $o_A[h'] = o_A[h]$), no new co-hypothesis is generated from B. In this case, the co-hypothesis list is kept as it is, i.e. $g[h'] = g[h]$. Only when the transition e outputs a non-epsilon symbol y ($o_A[e] = y \neq \varepsilon$ and $o_A[h'] \neq o_A[h]$), a new co-hypothesis f' is generated for each existing co-hypothesis f in $g[h]$ by adding a transition r, which originates from the state $n_B[f]$ with an input symbol y. The weight of f' can be calculated as $w_B[f'] = w_B[f] + w_B[r]$. New co-hypotheses generated as above are then stored in $g[h']$. We then use the following modified weight, instead of the pure weight $w_A[h']$, in the Viterbi search on A:

$$\alpha(h') = w_A[h'] + \min_{f' \in g[h']} w_B[f'] \qquad (8)$$

Table 1. Performance of one-pass decoding methods

	3g-1p	3g+Rnn-1p			
Max cohyp. list size	-	15	10	5	1
Word error rate [%]	26.8	24.5	24.5	24.5	**24.5**
Real-time factor	0.38	0.80	0.69	0.61	**0.51**

When the Viterbi search chooses the best hypothesis from different hypotheses that meet at the same state in A, their co-hypothesis lists are merged into one list.

For one-pass decoding with an RNNLM, we assume that WFST A is a fully composed n-gram based transducer, namely, $HCLG$, and WFST B is generated on-the-fly from the RNNLM using Algorithm 1. With this method, the exponential growth of states by RNNLM can be suppressed efficiently by limiting the size of co-hypothesis list $g[h]$. However, to save memory usage, it is preferable to erase all the states of the RNNLM WFST after recognizing each utterance. In addition, since $HCLG$ already contains n-gram probabilities, they must be canceled from the RNNLM WFST, i.e., $P'_{\rm comb}(w|p) = P_{\rm comb}(w|p)/P_{\rm ngram}(w|\phi(p))$ must be used in Algorithm 1.

5. EXPERIMENTS

We evaluated our decoding approach with the MIT lecture transcription task [17]. The training set consisted of 104 lectures of approximately 116.2 hours. The development and test sets were two and eight lectures of 2.0 and 7.8 hours, respectively.

We used tied-state triphone HMMs, a back-off trigram language model, and a class-based RNNLM for speech recognition. The feature vectors had 39 elements consisting of 12 MFCCs plus energy, and their delta and delta-delta components. The HMMs had 2,546 states and 32 Gaussians per state, which were made using lectures and seminars in the corpus. To train the HMMs, we used the differential maximum mutual information (dMMI) criterion [18, 19], which is equivalent to the minimum phone error (MPE) criterion.

The trigram model was trained using the transcriptions of 1.1 M words in the corpus. The vocabulary size of the lexicon was 47,448. We used Kneser-Ney smoothing [20] for backing off the trigram probabilities.

The class-based RNNLM had 250 word classes, where the classes were decided by the method in [3]. The hidden layer had 300 units. Accordingly, the input layer consisted of 47,488 + 300 units, and the output layer consisted of 47,488 + 250 units. We also inserted a compression layer of 60 units to reduce the computation of the RNN-based probabilities [3]. We did not observe any degradation of word accuracy by incorporating the compression layer in our preliminary experiment. The RNNLM was trained with the RNNLM toolkit [21], where the back-propagation through time (BPTT) technique was used in consideration of the dependence over multiple utterances. For handling this dependence in decoding, we modified the decoder so that the hidden layer activation vector of the best final state of RNNLM WFST was taken over to the initial state for the next utterance, as used in an N-best rescoring technique [5]. The test-set perplexities of the trigram model, the RNNLM, and their linear combination were 199, 253 and 157, respectively.

First we examined the trigram-based one-pass decoding (3g-1p) to obtain the baseline performance and then evaluated the proposed method using a linearly combined trigram/RNNLM (3g+Rnn-1p) while changing the maximum co-hypothesis list size. We used a computer including Intel Xeon X5570 2.54 GHz processors with a single thread to measure the decoder speed.

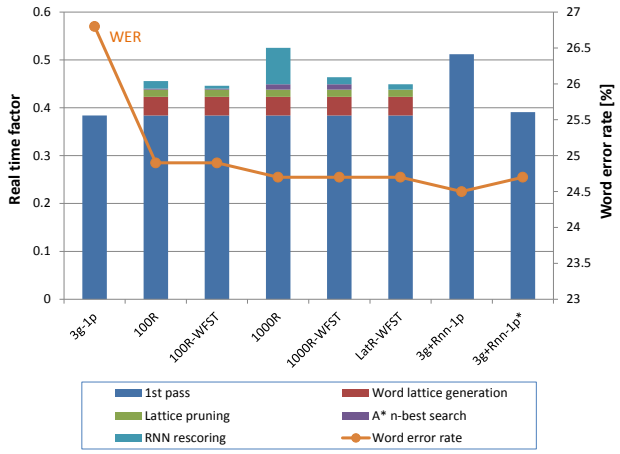

Fig. 2. WER and RTF of different decoding approaches with RNNLM.

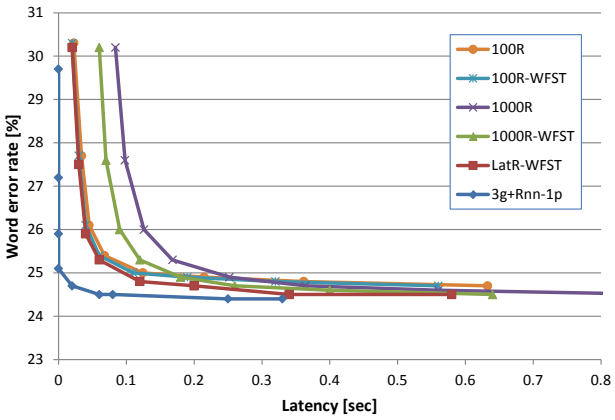

Fig. 3. Latency and WER of different decoding approaches with RNNLM

Table 1 shows the word error rate (WER) and the real-time factor (RTF) for each decoding condition, where the beam width was chosen for both methods, at which the WER of 3g-1p was almost saturated when we gradually increased the beam width.

We obtained a 26.8% WER and a 0.38 RTF for 3g-1p and a 24.5% WER and a 0.51 RTF for 3g+Rnn-1p with a max cohyp. list size of 1. During the 3g+Rnn-1p decoding, each partial hypothesis generated from the first WFST, i.e., $HCLG$, held multiple histories from the beginning of the speech. These histories were used to rescore the hypothesis by the maximum score over the histories when the hypothesis was extended by a new word. The max cohyp. list size is used to restrict each list.

When we used 15 for the list size, the RTF was 0.8. This is almost two times slower than the 3g-1p. However, the WER did not increase at all, and the RTF decreased to 0.51 even if we reduced the list size to 1. This implies that keeping at most one best history is adequate for each hypothesis conditioned by the previous two words, i.e., in trigram cases. In all the succeeding experiments, we set the list size to 1.

Second, we compared our decoding approach with N-best rescoring. Fig. 2 shows the RTF and WER for each method, where the decoding time is divided into 1st-pass decoding, word lattice generation [22], lattice pruning, A* N-best search, and RNNLM rescoring, all of which are necessary steps for N-best rescoring. Lattice pruning is performed based on the posterior probability of each lattice arc. The pruning threshold was chosen so that the WER did not increase.

We tested two types of N-best rescoring by the combined trigram/RNNLM. One is the standard method by which the sentence hypotheses in the N-best list are individually rescored, and the best hypothesis is chosen based on the rescored scores. The other is a more efficient method. An N-best list can be represented as a WFST consisting of linear transition chains that share one initial state. We combined the WFSTs of N-best list and RNNLM by a composition operation and found the best hypothesis from the composed WFST using a shortest-path algorithm. With this method, we removed the duplication of the RNN computation for the common prefix of the N-best list. This technique is expected to perform the rescoring step with almost the same time as fast N-best rescoring techniques such as an RNN probability cache [5] and a prefix tree based N-best list [6]. We also performed lattice rescoring with the RNNLM WFST by the composition and the shortest-path algorithm, where we used a similar technique to set the max co-hypothesis list size to 1.

RTFs and WERs in 100-best and 1000-best rescoring (100R and 1000R), and those with WFST operations (100R-WFST and 1000R-WFST), and lattice rescoring (LatR-WFST) are shown in Fig. 2 with 3g-1p and the proposed 3g+Rnn-1p. Although 3g+Rnn-1p needed a comparable decoding time with 1000R, it achieved the lowest WER. When we used a slightly narrower beam width (3g+Rnn-1p*), the RTF became smaller than those of other rescoring methods and comparable to 3g-1p, where the WER slightly increased from 24.5% to 24.7% but it equaled that of 1000R, 1000R-WFST, and LatR-WFST.

Finally, we show the performance of the proposed method in on-line processing, which we want to emphasize the most in this paper. We measured the latency (wait time) from the end of each utterance to the output of the recognition result. In this experiment, we simulated an on-line condition under which frame-by-frame processing does not proceed beyond the utterance time. Fig. 3 shows the WER and the latency of the one-pass and the two-pass approaches when we changed the beam width. In the two-pass approaches, computations excluding the 1st pass cannot start before the end of each utterance. Therefore, all additional computations for the 2nd-pass are included in the latency. As shown in Fig. 3, our proposed method reduced the latency by a factor of 10 compared to the other two-pass methods at the same WER. In addition, we emphasize again that this one-pass decoding not only reduces the latency but also decides the recognition result earlier without waiting for the end of each utterance, which is an important option for on-line applications.

6. CONCLUSION

In this paper, we proposed an efficient one-pass decoding method for real-time speech recognition employing a recurrent neural network language model (RNNLM). Our proposed method enabled one-pass Viterbi decoding with an RNNLM without approximation, where the RNNLM was represented as a prefix tree of possible word sequences, but only the part needed for decoding was generated on-the-fly and used to rescore each hypothesis using the on-the-fly composition technique that we previously proposed.

From the experimental results on the MIT lecture transcription task, we showed that the proposed method enabled one-pass decoding with small overhead for the RNNLM and achieved a slightly higher accuracy than 1000-best rescoring. The method also reduced the latency in two-pass decoding by a factor of 10.

Future work will include evaluation of our decoding method with a wide variety of tasks and extension to on-line adaptation of RNNLM in real-time processing.

7. REFERENCES

[1] H. Schwenk, "Continuous space language models," *Computer Speech and Language*, vol. 21, no. 3, pp. 492–518, 2007.

[2] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, , and S. Khudanpur, "Recurrent neural network based language model," in *Proc. Interspeech*, 2010, pp. 1045–1048.

[3] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Proc. ICASSP*, 2011, pp. 5528–5531.

[4] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocky, "Empirical evaluation and combination of advanced language modeling techniques," in *Proc. Interspeech*, 2011, pp. 605–608.

[5] S. Kombrink, T. Mikolov, M. Karafiat, and L. Burget, "Recurrent neural network based language modeling in meeting recognition," in *Proc. Interspeech*, 2011, pp. 2877–2880.

[6] Y. Si, Q. Zhang, T. Li, J. Pan, and Y. Yan, "Prefix tree based N-best list re-scoring for recurrent neural network language model used in speech recognition system," in *Proc. Interspeech*, 2013, pp. 3419–3423.

[7] A. Deoras, T. Mikolov, and K. Church, "A fast re-scoring strategy to capture long-distance dependencies," in *Proc. EMNLP*, 2011, pp. 1116–1127.

[8] M. Saraclar, M. Riley, E. Bocchieri, and V. Goffin, "Towards automatic closed captioning: low latency real time broadcast news transcription," in *Proc. ICSLP*, 2002, pp. 1741–1744.

[9] T. Hori, S. Araki, T. Yoshioka, M. Fujimoto, S. Watanabe, T. Oba, A. Ogawa, K. Otsuka, D. Mikami, K. Kinoshita, T. Nakatani, A. Nakamura, and J. Yamato, "Low-latency real-time meeting recognition and understanding using distant microphones and omni-directional camera," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 499–513, 2012.

[10] G. Lecorve and P. Motlicek, "Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition," in *Proc. Interspeech*, 2012.

[11] T. Hori, C. Hori, and Y. Minami, "Fast on-the-fly composition for weighted finite-state transducers in 1.8 million-word vocabulary continuous speech recognition," in *Proc. Interspeech2004–ICSLP*, 2004, vol. 1, pp. 289–292.

[12] T. Hori and A. Nakamura, "Generalized fast on-the-fly composition algorithm for WFST-based speech recognition," in *Proc. Interspeech2005–Eurospeech*, 2005, pp. 557–560.

[13] T. Hori, C. Hori, Y. Minami, and A. Nakamura, "Efficient WFST-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 4, pp. 1352–1365, 2007.

[14] D. Caseiro and I. Trancoso, "Transducer composition for "on-the-fly" lexicon and language model integration," in *Proc. ASRU*, 2001, pp. 393–396.

[15] C. Allauzen, M. Riley, and J. Schalkwyk, "A generalized composition algorithm for weighted finite-state transducers," in *Proc. Interspeech*, 2009, pp. 1203–1206.

[16] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 16, pp. 69–88, 2002.

[17] J. Glass, T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay, "Recent progress in the MIT spoken lecture processing project," in *Proc. Interspeech*, 2007, pp. 2553–2556.

[18] A. Nakamura, E. McDermott, S. Watanabe, and S. Katagiri, "A unified view for discriminative objective functions based on negative exponential of difference measure between strings," in *Proc. ICASSP*, 2009, pp. 1633–1636.

[19] E. McDermott, S. Watanabe, and A. Nakamura, "Discriminative training based on an integrated view of MPE and MMI in margin and error space," in *Proc. ICASSP*, 2010, pp. 4894–4897.

[20] R. Kneser and H. Ney, "Improved backing-off for M-gram language modeling," in *Proc. ICASSP'95*, 1995, vol. 1, pp. 181–184.

[21] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Cernocky, "RNNLM - Recurrent neural network language modeling toolkit," in *Proc. ASRU demo*, 2011, pp. 196–201.

[22] A. Ljolje, F. Pereira, and M. Riley, "Efficient general lattice generation and rescoring," in *Proc. Eurospeech*, 1999, pp. 1251–1254.