TRAINING DATA PSEUDO-SHUFFLING AND DIRECT DECODING FRAMEWORK FOR RECURRENT NEURAL NETWORK BASED ACOUSTIC MODELING

Naoyuki Kanda, Mitsuyoshi Tachimori, Xugang Lu, Hisashi Kawai

National Institute of Information and Communications Technology, Japan
{naoyuki.kanda,mitsuyoshi.tachimori,xugang.lu,hisashi.kawai}@nict.go.jp

ABSTRACT

We propose two techniques to enhance the performance of recurrent neural network (RNN)-based acoustic models. The first technique addresses training efficiency. Because RNNs require sequential input, it is difficult to randomly shuffle training samples to accelerate stochastic gradient descent based training. We propose a "pseudo-shuffling" procedure that instead augments training sample unexpectedness by skipping successive samples. The second proposed technique is a novel "direct decoding" framework in which the posterior probability of the RNN is inputted into a decoder without conversion into a hidden Markov model emission probability. In our large vocabulary speech recognition experiments with English lecture recordings, the first technique significantly improved RNN training efficiency, showing a 14.3% relative word error rate (WER) improvement. The second technique further achieved an additional 3.1% relative WER improvement. Our sigmoid-type RNN achieved a 10.7% better WER than same-sized deep neural networks without using long short-term memory cells.

Index Terms— Speech recognition, recurrent neural network, acoustic model

1. INTRODUCTION

Traditional speech recognition systems are based on the hidden Markov model (HMM), in which an emission probability $P(x_t|s_t)$ of observation x_t given an HMM state s_t at each time frame t is represented (see Fig. 1(a)). Before deep neural networks (DNNs) gained much attention, the emission probability was represented by Gaussian mixture models (GMMs). Recently, the DNN-HMM hybrid framework, in which DNNs are used to represent the emission probability instead of GMMs, was proposed, showing much better recognition accuracy than GMM-based acoustic models (AMs). [1, 2]. Because DNNs originally represent the posterior probability $P(s_t|x_t)$, the emission probability is estimated by applying a Bayes conversion in the DNN-HMM hybrid framework. The DNN-HMM hybrid framework has been applied in many scenarios [3, 4, 5, 6] and has achieved state-of-the-art recognition accuracy.

Recently, recurrent neural networks (RNNs) have been attracting much attention for AMs. Because RNNs are able to represent long-term time dependencies that DNNs cannot represent, RNNs seem to be a promising approach to improve AMs. However, using RNNs is not particularly easy. Simple sigmoid or tanh-type RNNs have often failed to produce better results than those obtained using state-of-the-art baseline systems [7, 8]. As far as we know, there are a few exceptions that report good results using simple RNNs. In [9], deep RNNs were trained using a modified backpropagation procedure. They reported a 4%–7% improvement using sigmoid-type RNNs; however, they compared RNNs with smaller DNNs.[1] No discussion was included to elucidate whether the RNNs could beat DNNs of the same size. Another study [10] proposed unfolded-type sigmoid RNNs; however, they also compared larger RNNs with smaller DNNs. Yet another study [7, 8, 11, 12] proposed the use of long short-term memory cells (LSTMs), showing better performance than DNNs. Although LSTMs appear to be promising, LSTMs normally entail higher computational cost than simple RNNs.[2] In this paper, we demonstrate that even a small sigmoid RNN has the ability to beat larger DNN-HMMs if the RNN-AM is appropriately trained and used.

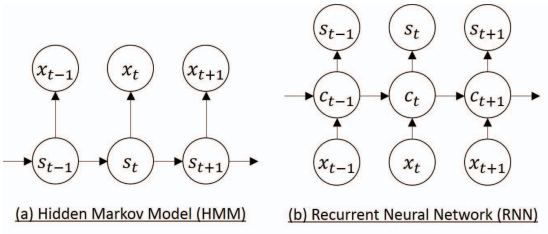

Fig. 1. *Model structure of (a) HMM and (b) RNN.*

One difficulty of RNNs originates in the training procedure. It is widely known that the neural network can be trained much more efficiently by the stochastic gradient decent if the training samples are randomly shuffled [13]. For example, a 7% relative improvement of the word error rate (WER) was reported by shuffling training samples [14]. On the other hand, training samples for RNNs must be a sequence, which makes the framewise shuffling of training samples impossible. As far as we have investigated, only a single paper [10] has tackled this problem; however, the authors proposed re-initializing the context information in unfolded RNNs in each frame to enable training samples to be shuffled. As a result, their model lost the ability to use infinite context information, which is a key ability of RNN models. In addition, their model could not be used in a "folded" structure, which causes decoding to need a much higher computational load. In contrast to the method above, our training method does not "shuffle" but does "augment the unexpectedness" of training samples by skipping successive training samples. We show that this simple idea achieves much better RNNs compared with the conventional training procedure.

Another difficulty with RNNs originates in the decoding procedure. Although RNNs have a strong ability to represent sequential structures, most previous studies that have used RNNs for AMs

[1] In [9], RNNs were initialized from the DNNs by assigning additional recurrent weights. Therefore, the RNNs were larger than the DNNs.

[2] In fairness to [8], the authors proposed a small representation of LSTMs and achieved good results.

[8, 9, 10, 11, 12] were based on the DNN-HMM hybrid framework,[3] in which dependencies in a state sequence are modeled by HMMs. In this paper, we propose a novel "direct decoding" framework in which the state dependency is modeled in the posterior probability of the RNNs. In the direct decoding framework, the posterior probability of the RNNs is inputted to a decoder without conversion into the frame-by-frame emission probability. In fact, the conventional DNN-HMM hybrid framework can be regarded as one configuration of this direct decoding framework. The direct decoding framework achieves a better WER, especially when the RNN-AMs are trained on the cross-entropy loss criterion.

The structure of this paper is as follows. In Section 2, an overview of RNN-based AMs is presented. In Section 3, the pseudo-shuffling procedure is proposed to efficiently train RNN-based AMs. In Section 4, we revisit the hybrid framework of DNN and HMM. We next describe the proposed direct decoding framework in Section 5. Finally, the experimental results for large vocabulary speech recognition using English lecture recordings are presented in Section 6.

2. RNN BASED ACOUSTIC MODELS

2.1. Model structure

Given an input sequence $\mathbf{x}_{1:T} = \{x_1, ..., x_T\}$, standard RNNs calculate the output $\mathbf{y}_{1:T} = \{y_1, ..., y_T\}$ by iteratively applying, from $t = 1$ to T, the following equations.

$$c_t = \mathcal{H}(U_{ih} \cdot x_t + U_{hh} \cdot c_{t-1} + b_h) \quad (1)$$

$$y_t = \mathcal{G}(U_{ho} \cdot c_t + b_o) \quad (2)$$

where U_{ih},U_{hh}, and U_{ho} respectively denote weight matrices within input–hidden, hidden–hidden, and hidden–output layers. Further, b_h and b_o respectively denote the bias vectors for the hidden and output layers, $\mathcal{H}$ denotes the activation function of a hidden layer that applies the sigmoid function in an element-wise fashion, and $\mathcal{G}$ is an output layer's softmax activation function.

When applying RNNs to AM, each node in the output layer is set to correspond to an HMM state. The i-th output of the RNN can then be regarded as the posterior probability of the i-th HMM state s_i given input history $\mathbf{x}_{1:t}$,

$$y_{t,i} = P(s_{t,i}|\mathbf{x}_{1:t}) \quad (3)$$

because the entire input history $\mathbf{x}_{1:t}$ is inputted recurrently to form the hidden layer's activation c_t (see the unfolded representation of RNN in Fig. 1(b)).

Recently, many researchers have proposed deep RNNs [8, 9, 11, 12], which consist of multiple recurrent layers, sometimes with additional non-recurrent layers. In this paper, we also use deep RNNs in which each hidden layer has a recurrent connection.

2.2. Truncated back propagation through time (BPTT)

RNNs are usually trained by using the truncated backpropagation through time (BPTT) procedure [17]. An overview and the pseudo code of this method are shown in Figs. 2(a) and 3(a), respectively. In this procedure, an RNN is first unfolded for T time steps, as shown in Fig. 2 (a) (in this case $T = 4$). Forward propagation (Eqs. 1 and 2) is then conducted to calculate predictions $y_t, ..., y_{t+T-1}$. Finally, errors for each prediction are back-propagated to calculate the gradient for updating RNNs. In the case of training based on the cross-entropy loss criterion, the error for prediction y_t is the difference between prediction y_t and reference ref_t. This procedure is repeated by shifting T training samples, as described in Fig. 3(a). It is noteworthy that c_{t-1} is essential to perform each iteration of the truncated BPTT procedure because of Eq. 1, which makes frame-wise shuffling of the training samples impossible.

[3] Exceptions are those which used RNNs as a feature extractor (known as the Tandem approach) [15] and those based on the connectionist temporal classification model [7, 16].

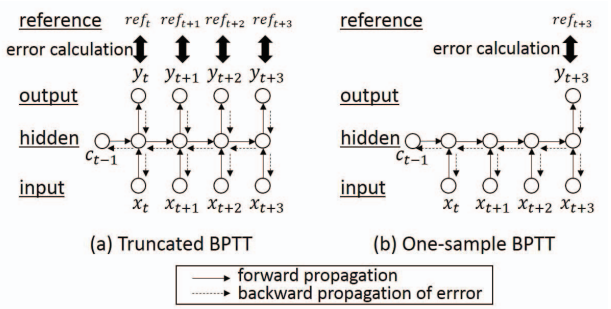

Fig. 2. *Overview of truncated BPTT and one-sample BPTT.*

(a) Truncated BPTT

```
for (t = 0 ; t < num_samples; t = t + T){
  RNN.truncated_BPTT(c_{t-1}, x_t, ..., x_{t+T-1}, ref_t, ..., ref_{t+T-1})
}
```

(b) One-sample BPTT

```
for (t = 0 ; t < num_samples; t = t + 1){
  RNN.one_sample_BPTT(c_{t-1}, x_t, ..., x_{t+T-1}, ref_{t+T-1})
}
```

(c) One-sample BPTT with pseudo-shuffling (= proposed method)

```
for (loop = 0 ; loop < T; loop = loop + 1){
  for (t = loop ; t < num_samples; t = t + T){
    RNN.one_sample_BPTT(c_{t-1}, x_t, ..., x_{t+T-1}, ref_{t+T-1})
  }
}
```

Fig. 3. *Pseudo code of various truncated BPTT procedures.*

3. ONE-SAMPLE BPTT WITH PSEUDO-SHUFFLING

3.1. One-sample BPTT

Before explaining the pseudo-shuffling procedure, we introduce "one-sample BPTT," which is a variant of truncated BPTT.[4] An overview and the pseudo code of this method are shown in Figs. 2(b) and 3(b), respectively. Contrary to the case of the original truncated BPTT, error is calculated only in the final prediction (in this case, y_{t+3}). This procedure is repeated by shifting samples one-by-one, as described in Fig. 3(b).

Compared with the original truncated BPTT, the one-sample BPTT has the advantage that the error calculated for each prediction must be delivered T-time back steps. On the contrary, in the case of original truncated BPTT, errors calculated for the NON-final

[4] Although this algorithm was sometimes called "truncated BPTT" [18], we refer to it as "one-sample BPTT" to discriminate it from the method described in Sec. 2.2

prediction (in the case of Fig. 2 (a), y_t, y_{t+1}, y_{t+2}) are delivered less than T-time back steps.

3.2. Pseudo-shuffling

It is widely known that the training efficiency of neural networks is heavily dependent on the order of training samples [13, 14]. As an extreme case, if the same M training samples are inputted as mini-batch data, the average gradient obtained by the M samples is exactly equal to that obtained by one sample. Therefore, in order to make training efficient, the order of training samples is usually randomized when training DNNs.

On the other hand, in the case of the original truncated BPTT, the method uses successive T training samples for one update (Fig. 2(a)). Even in the case of the one-sample BPTT, successive training samples are repeatedly inputted to the RNN (Fig. 3(b)). Those successive samples could cause training RNNs to be highly inefficient because samples from adjacent frames of the speech stream are often very similar to each other.

To improve the BPTT procedure, we propose a pseudo-shuffling procedure with one-sample BPTT. The pseudo code of this method is described in Fig. 3(c). The differences between this method with and without pseudo-shuffling are indicated in red. In the pseudo-shuffling procedure, the training samples for BPTT are selected by skipping T samples. By looping T times, all the training samples are traversed. Using this procedure, the RNN is trained on at least T-frame different training samples for each iteration of training. This procedure does not randomize the order of training samples, but instead "augments the unexpectedness" of training samples. We experimentally show the superiority of this method in Section 6.

3.3. Mini-batch processing for RNN training

The mini-batch processing makes neural network training much more efficient, especially when the training is done on GPGPU machines. In the case of DNNs, making a mini-batch of B samples is very easy: we just randomly select B training samples. On the contrary, in the case of RNN training, we must keep the sequential nature of the training samples. Therefore, some scheme is needed to realize mini-batch training for RNNs.

In this paper, we use the on-the-fly shuffling scheme [19] in which a mini-batch consists of randomly lined-up B sequences of training samples (see Fig. 2 in [19]). Note that if we use conventional truncated BPTT with mini-batch processing, the mini-batch size B should be smaller (e.g., $B = 16$) than the case of DNN training (e.g., $B = 256$), because the number of training samples observed for one update is $B \cdot T$. In the case of one-sample BPTT, the number of training samples for one update is B, so we can use the same value as used for DNN training.

It is noteworthy that the on-the-fly shuffling procedure does not solve the problem of using successive training samples, which we mentioned in the previous section. Although on-the-fly shuffling achieved a nice improvement in WER in its original paper [19], our experiment revealed that the on-the-fly shuffling is not enough to obtain good RNN.

4. DNN-HMM HYBRID FRAMEWORK

We now move on to the topic of a decoding procedure. Before introducing our proposed framework, we first review the conventional DNN-HMM hybrid framework.

4.1. Overview of the DNN-HMM hybrid framework

Traditional speech recognition systems are based on the noisy channel model with HMM-based AMs. In this framework, a word sequence W, given observations $\mathbf{x}_{1:T}$, is estimated as follows.

$$\tilde{W} = \arg\max_W P(W|\mathbf{x}_{1:T}) \quad (4)$$

$$= \arg\max_W \frac{P(\mathbf{x}_{1:T}|W)P(W)}{P(\mathbf{x}_{1:T})} \quad (5)$$

$$\simeq \arg\max_W \frac{P(\mathbf{x}_{1:T}|\mathbf{s}_{1:T})P(\mathbf{s}_{1:T}|W)P(W)}{P(\mathbf{x}_{1:T})}. \quad (6)$$

In Eq. 6, $\mathbf{s}_{1:T} = \{s_1, ..., s_T\}$ denotes a sequence of HMM states and Viterbi approximation is used. Term $P(\mathbf{x}_{1:T}|\mathbf{s}_{1:T})$ denotes the probability of emitting observations $\mathbf{x}_{1:T}$ from $\mathbf{s}_{1:T}$. According to the structure of HMM, x_t only depends on s_t (see Fig. 1(a)). Therefore $P(\mathbf{x}_{1:T}|\mathbf{s}_{1:T})$ can be calculated as a product of the emission probability $P(x_t|s_t)$ as

$$P(\mathbf{x}_{1:T}|\mathbf{s}_{1:T}) = \prod_{t=1}^{T} P(x_t|s_t). \quad (7)$$

Before DNN-AMs gathered much attention, $P(x_t|s_t)$ was usually calculated using GMMs. For the remaining terms in Eq. 6, $P(\mathbf{s}_{1:T}|W)$ is calculated as a product of the HMM's transition probability and word pronunciation probability, and $P(W)$ is calculated using a language model. Because $P(\mathbf{x}_{1:T})$ is constant for each hypothesis, it is ignored in the argmax calculation.

In the DNN-HMM hybrid framework [1, 2], emission probability $P(x_t|s_t)$ is calculated using DNNs instead of GMMs. Because the DNNs originally represent the posterior probability $P(s_t|x_t)$, emission probability $P(x_t|s_t)$ is calculated using Bayes' rule as

$$P(x_t|s_t) = \frac{P(s_t|x_t)P(x_t)}{P(s_t)} \propto \frac{P(s_t|x_t)}{P(s_t)}. \quad (8)$$

Here, $P(s_t)$ can be estimated by counting the number of states in the aligned training data. Because $P(x_t)$ is constant for each HMM state, it is ignored in the argmax calculation.

4.2. Problem of the combination of RNN and HMM

In the previous studies [8, 9, 10, 11, 12], RNN outputs $P(s_t|\mathbf{x}_{1:t})$ are also divided by $P(s_t)$, as they are in the case of DNN-HMMs. Here, it is noteworthy that if we divide $P(s_t|\mathbf{x}_{1:t})$ by $P(s_t)$, we obtain a value that is proportional to $P(\mathbf{x}_{1:t}|s_t)$ instead of $P(x_t|s_t)$, as follows.

$$\frac{P(s_t|\mathbf{x}_{1:t})}{P(s_t)} = \frac{P(\mathbf{x}_{1:t}|s_t)}{P(\mathbf{x}_{1:t})} \propto P(\mathbf{x}_{1:t}|s_t). \quad (9)$$

First, $P(\mathbf{x}_{1:t}|s_t)$ is no longer proportional to $P(x_t|s_t)$ because the previous input history $\mathbf{x}_{1:t-1}$ and s_t have strong dependency in RNNs. Therefore, this value cannot be used as-is in Eq. 7. A practical trick to deal with $P(\mathbf{x}_{1:t}|s_t)$ in decoding is the following: Let $\hat{\mathbf{x}}_{1:T} = \{\hat{x}_1, ..., \hat{x}_T\}$ be a sequence in which $\hat{x}_t$ is defined as $\hat{x}_t = \mathbf{x}_{1:t}$. We then reconsider the recognition problem as a problem to estimate word sequence W given observations $\hat{\mathbf{x}}_{1:T}$. This trick works well in practice. However, it is worth noting that the RNN has the power to estimate not only the current emission probability $P(\hat{x}_t|s_t) = P(\mathbf{x}_{1:t}|s_t)$ but also the emission probability of the previous observation $P(\hat{x}_{t-1}|s_t)$, as shown below.

$$P(\hat{x}_{t-1}|s_t) = P(\mathbf{x}_{1:t-1}|s_t) = \int_{x_t} P(\mathbf{x}_{1:t}|s_t)dx_t. \quad (10)$$

Unfortunately, HMM has no means to incorporate $P(\hat{x}_{t-1}|s_t)$, even if this value is already calculated.[5] Therefore, the power of RNNs cannot be used fully in the DNN-HMM framework. This problem occurs as long as the framework is based on state-based emission probability.

5. DIRECT DECODING FRAMEWORK

5.1. Overview of the direct decoding framework

Instead of using the DNN-HMM hybrid framework, we propose a "direct decoding" framework in which the RNN outputs are used without frame-by-frame Bayes conversions. The direct decoding framework is represented as follows.

$$\tilde{W} = \arg\max_W P(W|\mathbf{x}_{1:T}) \quad (11)$$

$$\simeq \arg\max_W P(W|\mathbf{s}_{1:T})P(\mathbf{s}_{1:T}|\mathbf{x}_{1:T}). \quad (12)$$

In Eq. 12, Viterbi approximation is used as in the conventional framework. In general, $P(\mathbf{s}_{1:T}|\mathbf{x}_{1:T})$ in the equation above is very difficult to use in decoding because the scores for all combinations of $\mathbf{s}_{1:T}$ in the recognition hypotheses need to be calculated. Therefore, instead of a strict estimation of $P(\mathbf{s}_{1:T}|\mathbf{x}_{1:t})$, we approximately estimate $P(\mathbf{s}_{1:T}|\mathbf{x}_{1:t})$ by using RNN outputs $P(s_t|\mathbf{x}_{1:t})$, as follows.

$$P(\mathbf{s}_{1:T}|\mathbf{x}_{1:T}) = \prod_{t=1}^{T} P(s_t|\mathbf{s}_{1:t-1}, \mathbf{x}_{1:T}) \quad (13)$$

$$\simeq \prod_{t=1}^{T} P(s_t|\mathbf{s}_{1:t-1}, \mathbf{x}_{1:t}) \quad (14)$$

$$\simeq \prod_{t=1}^{T} P(s_t|\mathbf{x}_{1:t}). \quad (15)$$

Equation 13 is derived from Bayes' rule and strictly holds. In Eq. 14 we assume that the posterior distribution of s_t can be well estimated without depending on future observations $\mathbf{x}_{(t+1):T}$. Generally, this assumption does not hold; however, we can expect this approximation to work when sufficient future information is included in x_t, which can be achieved by splicing the input feature or delaying the reference label.[6] Finally, in Eq. 15, we assume that the posterior probability of $P(s_t|\mathbf{s}_{1:t-1}, \mathbf{x}_{1:t})$ can be well approximated by the outputs of RNN $P(s_t|\mathbf{x}_{1:t})$. It is important to note that this approximation does not equal the assumption of the independence between s_t and $\mathbf{s}_{1:t-1}$. Even when s_t and $\mathbf{s}_{1:t-1}$ are strongly dependent, one can expect this approximation to work well if an RNN has a strong power to estimate s_t from observation $\mathbf{x}_{1:t}$ alone. Although this approximation is a coarse one, experimentally, it works.

The remaining term in Eq. 12, $P(W|\mathbf{s}_{1:T})$, is calculated as

$$P(W|\mathbf{s}_{1:T}) = \frac{P(\mathbf{s}_{1:T}|W)P(W)}{P(\mathbf{s}_{1:T})}. \quad (16)$$

Here, $P(\mathbf{s}_{1:T}|W)$ and $P(W)$ are calculated in the same manner as in the HMM-based framework. Term $P(\mathbf{s}_{1:T})$ can be estimated from the state sequence in aligned training data using conventional language modeling tools. In this study, we used a bigram model to calculate $P(\mathbf{s}_{1:T})$, as shown below.

$$P(\mathbf{s}_{1:T}) \simeq \prod_{t=1}^{T} P(s_t|s_{t-1}). \quad (17)$$

5.2. Relation to the DNN-HMM hybrid framework

By inputting Eq. 16 into Eq. 12, one can obtain

$$\tilde{W} \simeq \arg\max_W \frac{P(\mathbf{s}_{1:T}|\mathbf{x}_{1:T})P(\mathbf{s}_{1:T}|W)P(W)}{P(\mathbf{s}_{1:T})}. \quad (18)$$

Comparing Eqs. 6 and 18, it is apparent that while $P(\mathbf{x}_{1:T}|\mathbf{s}_{1:T})$ is calculated by Eqs. 7 and 8 in the DNN-HMM framework, $P(\mathbf{s}_{1:T}|\mathbf{x}_{1:T})/P(\mathbf{s}_{1:T})$ is calculated in the direct decoding framework. Recall that the DNN outputs are divided by $P(s_t)$ frame-by-frame in the DNN-HMM framework (Eq. 8). Hence, the difference between the DNN-HMM and direct decoding frameworks can be summarized by the method used to calculate $P(\mathbf{s}_{1:T})$. In fact, if we use a unigram probability $P(s_t)$ in Eq. 17, then results based on the direct decoding framework become identical to those based on the DNN-HMM hybrid framework. Therefore, the DNN-HMM hybrid framework can be regarded as one configuration of the direct decoding framework.

5.3. Implementational issues

We modified our decoder to enable the calculation of $P(s_t|s_{t-1})$ for Eq. 17 when expanding the recognition hypotheses. In our implementation, each hypothesis contains the previous HMM-states, and the decoder evaluates each hypothesis according to Eq. 12 $\sim$ Eq. 17. We would like to point out that if the decoder is based on the weighted finite state transducer (WFST) $H \circ C \circ L \circ G$, which converts the HMM-state sequence into a word sequence, the same thing can be done by composing S^{-1} with $H \circ C \circ L \circ G$.[7] Here, S^{-1} is a WFST from an HMM-state sequence to the same HMM-state sequence in which each arc has probability $P(s_t|s_{t-1})^{-1}$.

5.4. Sequence discriminative training for the direct decoding framework

It is widely known that DNNs trained using a sequence discriminative criterion (e.g., the MMI or sMBR criteria) achieve better WER than those trained with a cross-entropy loss criterion [14, 22]. The original form of the MMI objective function for DNN training is described as follows.

$$\mathcal{F}^{MMI}(\theta) = \sum_u \log \frac{P(\mathbf{x}_{1:T}|\mathbf{s}_{1:T})^\kappa P(\mathbf{s}_{1:T}|W)}{\sum_{W'} P(\mathbf{x}_{1:T}|\mathbf{s}_{1:T})^\kappa P(\mathbf{s}_{1:T}|W')} \quad (19)$$

Here, u indicates the utterance ID, θ indicates the parameters to be updated, and κ is the acoustic scaling parameter.

If we used the direct decoding framework, this equation also should be modified to meet the framework. In this case, the MMI objective function for network training becomes

$$\mathcal{F}^{MMI}(\theta) = \sum_u \log \frac{P(W|\mathbf{s}_{1:t})P(\mathbf{s}_{1:T}|\mathbf{x}_{1:T})^\kappa}{\sum_{W'} P(W'|\mathbf{s}_{1:T})P(\mathbf{s}_{1:T}|\mathbf{x}_{1:T})^\kappa} \quad (20)$$

In practice, this modification is done by replacing the DNN-HMM-framework-based lattice generation procedure with the direct-decoding-based one.

[5] Practically, it is not easy to calculate, but theoretically, it can be.

[6] This assumption can be completely removed by using bi-directional models [20], which will be explored in our future work.

[7] That was not done for this study because our current decoder is based on $C \circ L \circ G$ in [21], and the authors had insufficient resources to test $H \circ C \circ L \circ G$-based decoding in that particular study.

Table 2. *Frame accuracy (FA) and word error rates (WER) of RNNs with different training procedures.*

Model type	Training method	BPTT step (=T)	On-the-fly shuffling(=B)	FA (%) on Dev set	WER (%) Dev set	WER (%) Tst2013	Number of parameters
RNN.512.3	Truncated BPTT	16	16	53.9	16.75	22.17	5.6M
	One-sample BPTT	9	256	56.2	16.40	21.94	
	One-sample BPTT w/ pseudo-shuffling	9	256	**62.0**	**14.10**	**18.83**	
RNN.512.5	One-sample BPTT w/ pseudo-shuffling	9	256	**62.6**	**13.77**	**17.96**	6.6M

Table 3. *Word error rates (WER) of RNNs with different decoding frameworks.*

Model type	Framework	N-gram calculation in Eq. 17	WER (%) Dev set	WER (%) Tst2013	Number of parameters
RNN.512.3	DNN-HMM Framework	$P(s_t)^{0.8}$	14.11	19.08	5.6M
		$P(s_t)^{1.0}$ (=baseline)	14.10	18.83	
		$P(s_t)^{1.2}$	14.32	18.84	
	Direct Decoding Framework	$P(s_t\|s_{t-1})^{1.0}$	14.31	20.12	
		$P(s_t\|s_{t-1})^{1.5}$	**13.59**	18.67	
		$P(s_t)^{0.75} \cdot P(s_t\|s_{t-1})^{0.75}$	13.70	**18.23**	
RNN.512.5	DNN-HMM Framework	$P(s_t)^{1.0}$ (=baseline)	13.77	17.96	6.6M
	Direct Decoding Framework	$P(s_t)^{0.75} \cdot P(s_t\|s_{t-1})^{0.75}$	**13.49**	**17.63**	

Table 1. *Frame accuracy (FA) and word error rates (WER) of DNNs.*

Model type	FA (%) on Dev set	WER (%) Dev set	WER (%) Tst2013	Number of parameters
DNN.512.5	50.2	15.36	20.41	5.6M
DNN.1024.5	52.4	13.80	18.71	13.3M

6. EXPERIMENTS

6.1. Evaluation settings

We evaluated our system based on the IWSLT 2013 ASR English test dataset (designated as Tst2013) [23], which includes 4.8 hours (28 speakers) of TED lecture recordings. As a development set to tune system parameters, we used 5.7 hours (29 speakers) of TED lecture recordings (a combination of the data called dev2010, tst2010, and dev2012).

As training data for the AMs, we used a 346-hour speech corpus, which comprised 163 hours of TED recordings, 102 hours of the HUB4 corpus, and 81 hours of the WSJ corpus. A DNN-based AM with five hidden layers, each of which had either 512 or 1,024 nodes, was trained as a baseline model. The output layer had 7,838 nodes that corresponded to context-dependent phone HMM states. As acoustic features, 72 dimension filter-bank features (24 filter-bank features, delta coefficients, and delta-delta coefficients) with mean and variance normalization per speaker were used. We concatenated the features of both the previous and following seven frames (15 frames of features in total) when inputting to DNNs. Each model was initialized using the discriminative pre-training method [24] and was fine-tuned using the standard stochastic gradient descent based on cross-entropy loss criterion. The initial learning rate[8] was set to 1.0 and the mini-batch size was set to 256. The learning rate was tuned during training using the development set. Training was iterated until the relative improvement of the frame accuracy of the development set became less than 0.1%.

We then trained RNN-AMs with three or five hidden layers, each of which had 512 nodes. In both RNN models, every hidden layer had a recurrent connection. The output layer had 7,838 nodes, which was the same as the DNN models. The 72 dimensional filter-bank features were used as acoustic features. When features were fed into the RNNs, the prior and subsequent three frames of features were concatenated. Additionally, we delayed the reference label by four frames. As a result, the RNNs could observe seven future frames to predict the reference label; configuration was used to be fair to the DNN models. We initialized the RNNs by using the sparse initialization technique [26] with no pre-training. The RNNs were trained using the method described in Section 3. The training methods are compared in the next section. The initial learning rate was set to 2.0 for the three-hidden-layer model and 1.0 for the five-hidden-layer model, and tuned using the development set. When training the MMI, networks were copied from the cross-entropy-trained models, and three epochs of training were conducted with a learning rate of 0.0025.

In addition, a bigram model of the HMM-state for Eq. 17 was estimated from the aligned training data. Through our preliminary evaluation, we found that the probability that is estimated by a language modeling toolkit tends to become too high for known (observed) sequences and too low for unknown sequences. As we divide the hypothesis score by $P(s_{1:T})$, too-high probabilities of $P(s_{1:T})$ for known sequences become too-low hypothesis scores for known sequences. To mitigate this problem, we used two smoothing methods that (1) added a constant value to each entry when estimating bigrams,[9] and (2) interpolated a bigram with unigram probability. We present the experimental results in the following subsection.

Finally, we trained a 4-gram language model with modified Kneser–Ney smoothing [28]. We first trained two models; one was trained based on 184K sentences of the TED transcription and another was trained based on selected 10M sentences from the English Gigaword corpus, fifth edition. These language models were then interpolated and used in the decoder. When decoding, the language model weight was tuned by the development set.

6.2. Evaluation of one-sample BPTT with pseudo-shuffling

We first evaluated the conventional DNN-based models. The results are presented in Table 1. In this table, DNN.x.y indicates a DNN

[8] Our training tool uses the mini-batch average of gradient instead of mini-batch sum of gradient, which some AM tools (e.g., Kaldi[25]) use. Therefore, the initial learning rate must be set relatively high.

[9] We used the SRILM toolkit [27] with the setting "-gt1max 0 -addsmooth2 5."

Table 4. *Word error rates (WER) of RNNs with MMI training*

Model type	Decoding framework	MMI training framework	WER (%)	
			Dev set	Tst2013
RNN.512.3	DNN-HMM	DNN-HMM (Eq. 19)	12.31	16.73
		Direct Decoding (Eq. 20)	12.71	17.92
	Direct Decoding	DNN-HMM (Eq. 19)	12.20	16.34
		Direct Decoding (Eq. 20)	12.25	16.65

with y hidden layers, each of which has x nodes. Although the big DNN with five hidden layers of 1,024 nodes achieved an 18.71% WER, the small DNN with 512 nodes produced much a worse WER of 20.41%. This result indicates the importance of the number of parameters for DNN models.

We then evaluated an RNN with three hidden layers with 512 nodes. This RNN had the same number of parameters as the smaller DNN (5.6M parameters). The frame accuracy (FA) and WER with various training methods are listed in the upper three columns of Table 2 (denoted as "RNN.512.3"). Note that we tested various combination of parameters (including the learning rate), and the most representative results are listed in the table. When the RNN was trained using the truncated BPTT, while the FA (53.9%) became better than DNN models, the WER (22.17% for Tst2013) was much worse than the DNNs.[10] An RNN trained by the one-sample BPTT slightly improved the FA and WER; however, the WER was still much worse than that of the DNNs. Finally, an RNN trained by the one-sample BPTT with pseudo-shuffling achieved a 5.8-point better FA (56.2% to 62.0%) and 3.11-point better WER (21.94% to 18.83%) for Tst2013. This WER was 1.58 percentage points better even than the same-sized DNN (20.41%). The difference between the one-sample BPTT with and without pseudo-shuffling was just the order of the training data used, as described in Section 3.2. These results indicate how important the sampling order is to train good RNNs.

We also trained an RNN with five hidden layers of 512 nodes with one-sample BPTT and pseudo-shuffling. Although this model had only 6.6M parameters, it achieved a much better WER (17.96%) even than the larger DNNs (18.71%) with 13.3M parameters.

6.3. Evaluation of the direct decoding framework

Before evaluating the direct decoding framework, we again evaluated the smaller RNN (three hidden layers, 512 nodes) based on the DNN-HMM framework. The results are listed in the upper part of Table 3. Note that using unigram $P(s_t)$ instead of bigram in Eq. 17 is equivalent to using the DNN-HMM framework, as we described in Section 5.2. In this experiment, we additionally investigated the results with various values of scaling parameter for $P(s_t)$, which is known as the prior smoothing [29]. As shown in the table, a prior smoothing of 0.8 or 1.2 produced no improvement, but instead degraded the WER. Therefore, we set a baseline with $P(s_t)^{1.0}$. In this case, the WER for Dev and Tst2013 was 14.10% and 18.83%, respectively.

We then evaluated the same RNN with the direct decoding framework. The results are shown in the middle of Table 3. When we used bigram $P(s_t|s_{t-1})$ with a scaling factor of 1.0, the WER was degraded to 14.31% and 20.12% for Dev and Tst2013, respectively. However, when we used a scaling factor of 1.5, the WER for Dev was significantly improved to 13.59%. The WER for Tst2013 was also improved to 18.67%, which is slightly better than that of the baseline DNN-HMM framework (18.84%). Finally, the logarithmic interpolation of unigram and bigram ($P(s_t)^{0.75} \cdot P(s_t|s_{t-1})^{0.75}$) gave us a further improvement for Tst2013, and the WER became 18.23%, which was 10.7% better than a DNN of the same size. The WER for Dev showed a slight degradation (13.70%); however, it was still significantly better than the case of the DNN-HMM framework (14.10%) and the score of the same-sized DNN (15.36%).

We also evaluated the larger RNN with five hidden layers of 512 nodes. The results are shown in the last two columns of Table 3. For this RNN, the direct decoding framework again achieved a better WER for both Dev and Tst2013. The final WER for Tst2013 was 17.63%, which was 1.08 points better than the much larger DNN with 13.3M parameters. These results indicate the superiority of the direct decoding framework.

Finally, we evaluated the effect of MMI training based on the direct decoding framework. In this experiment, we used a setting of $P(s_t)^{0.75} \cdot P(s_t|s_{t-1})^{0.75}$. The results are listed in Table 4. Here, we used an RNN with three hidden layers of 512 nodes. First, when the decoding framework was the DNN-HMM framework, an RNN trained by a "DNN-HMM based MMI-criterion" (Eq. 19) showed a significant improvement, and achieved 12.31% and 16.73% WERs for Dev and Tst2013, respectively. In contrast, an RNN trained by a "direct-decoding-based MMI-criterion" (Eq. 20) showed much worse results: 12.71% and 17.92% WERs for Dev and Tst2013, respectively. These results were within our expectations because there was a mismatch of the training and decoding criteria. When the decoding framework was the direct decoding framework, an RNN trained by a direct-decoding based MMI-criterion showed slightly better results than those of the DNN-HMM case: 12.25% and 16.65% WERs for Dev and Tst2013, respectively. Contrary to our expectations, the best results were obtained by an RNN trained by a DNN-HMM based MMI-criterion with the direct decoding framework (12.20% and 16.34% WERs for Dev and Tst2013, respectively). We did not expect this result because there was a mismatch between the training and decoding criterion. Note that we used a unigram language model for the MMI training, which was different from the language model that we used for decoding. We speculate this mismatch might have caused the unexpected behavior above. In either case, the direct decoding framework showed better results than the DNN-HMM framework, even when the model was trained based on the MMI criterion.

7. CONCLUSION

This paper proposed two techniques to enhance the performance of RNN-AMs. The first is the one-sample BPTT procedure with pseudo-shuffling that accelerates training efficiency by augmenting the unexpectedness of training samples. The second is the direct decoding framework for RNN-AMs, which uses RNN outputs without frame-by-frame Bayes conversion. Both methods significantly improved the performance of RNN-AMs, and our small RNN-AMs finally achieved much better results than the larger DNN-based AMs, without using complicated network structures like LSTMs.

[10] Note that it is often observed that a model with better FA produces worse WER, especially when comparing different types of networks.

8. REFERENCES

[1] Frank Seide, Gang Li, and Dong Yu, "Conversational speech transcription using context-dependent deep neural networks.," in *Proc. INTERSPEECH*, 2011, pp. 437–440.

[2] George E Dahl, Dong Yu, Li Deng, and Alex Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and Brian Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[4] Naoyuki Kanda, Ryu Takeda, and Yasunari Obuchi, "Elastic spectral distortion for low resource speech recognition with deep neural networks," in *Proc. ASRU*, 2013, pp. 309–314.

[5] G Heigold, V Vanhoucke, A Senior, P Nguyen, M Ranzato, M Devin, and J Dean, "Multilingual acoustic models using distributed deep neural networks," in *Proc. ICASSP*, 2013, pp. 8619–8623.

[6] Peng Shen, Xugang Lu, Naoyuki Kanda, Masahiro Saiko, and Chiori Hori, "The NICT ASR system for IWSLT 2014," in *Proceedings of IWSLT*, 2014, pp. 113–118.

[7] Alex Graves, A-R Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. ICASSP*. IEEE, 2013, pp. 6645–6649.

[8] Haşim Sak, Andrew Senior, and Françoise Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv e-prints*, 2014.

[9] Chao Weng, Dong Yu, Shinji Watanabe, and Biing-Hwang Fred Juang, "Recurrent deep neural networks for robust speech recognition," in *Proc. ICASSP*. IEEE, 2014, pp. 5532–5536.

[10] George Saon, Hagen Soltau, Ahmad Emami, and Michael Picheny, "Unfolded recurrent neural networks for speech recognition," in *Proc. INTERSPEECH*, 2014.

[11] Hasim Sak, Oriol Vinyals, Georg Heigold, Andrew Senior, Erik McDermott, Rajat Monga, and Mark Mao, "Sequence discriminative distributed training of long short-term memory recurrent neural networks," *entropy*, vol. 15, no. 16, pp. 17–18, 2014.

[12] Jürgen T Geiger, Zixing Zhang, Felix Weninger, Björn Schuller, and Gerhard Rigoll, "Robust speech recognition using long short-term memory recurrent neural networks for hybrid acoustic modelling," in *Proc. INTERSPEECH*, 2014.

[13] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

[14] Hang Su, Gang Li, Dong Yu, and Frank Seide, "Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription.," in *Proc. ICASSP*, 2013, pp. 6664–6668.

[15] Oriol Vinyals, Suman V Ravuri, and Daniel Povey, "Revisiting recurrent neural networks for robust ASR," in *Proc. ICASSP*. IEEE, 2012, pp. 4085–4088.

[16] Alex Graves and Navdeep Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1764–1772.

[17] Herbert Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, GMD-Forschungszentrum Informationstechnik, 2002.

[18] Mikael Boden, "A guide to recurrent neural networks and backpropagation," 2001.

[19] Georg Heigold, Erik McDermott, Vincent Vanhoucke, Andrew Senior, and Michiel Bacchiani, "Asynchronous stochastic optimization for sequence training of deep neural networks," in *Proc. ICASSP*. IEEE, 2014, pp. 5587–5591.

[20] Mike Schuster and Kuldip K Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[21] Paul Richard Dixon, Chiori Hori, and Hideki Kashioka, "Development of the SprinTra WFST speech decoder," *Journal of the National Institute of Information and Communications Technology Vol*, vol. 59, no. 3/4, 2012.

[22] Karel Veselỳ, Arnab Ghoshal, Lukás Burget, and Daniel Povey, "Sequence-discriminative training of deep neural networks.," in *Proc. INTERSPEECH*, 2013, pp. 2345–2349.

[23] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico, "Report on the 10th IWSLT evaluation campaign," in *Proc. IWSLT*, 2013, pp. 29–38.

[24] Frank Seide, Gang Li, Xie Chen, and Dong Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. ASRU*. IEEE, 2011, pp. 24–29.

[25] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondřej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, et al., "The Kaldi speech recognition toolkit," in *Proc. ASRU*. IEEE, 2011.

[26] James Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 735–742.

[27] Andreas Stolcke et al., "SRILM-an extensible language modeling toolkit.," in *INTERSPEECH*, 2002.

[28] Stanley F Chen and Joshua Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–393, 1999.

[29] Navdeep Jaitly, Patrick Nguyen, Andrew Senior, and Vincent Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition.," in *INTERSPEECH*, 2012.