Ph.D. DISSERTATION

End-to-End Online Speech Recognition with Recurrent Neural Networks

재귀형 인공신경망을 이용한 온라인 음성인식

BY

KYUYEON HWANG

FEBRUARY 2017

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Abstract

Recurrent neural networks (RNNs) have shown outstanding sequence to sequence modeling performance. Thanks to recent advances in end-to-end training approaches for automatic speech recognition (ASR), RNNs can learn direct mapping functions from the sequence of audio features to the sequence of output characters or words without any intermediate phoneme or lexicon layers. So far, majority of studies on end-to-end ASR have been focused on increasing the accuracy of speech recognition to the level of traditional state-of-the-art models. However, although the end-to-end ASR models have reached the accuracy of the traditional systems, their application has usually been limited to utterance-level speech recognition with pre-segmented audio instead of online speech recognition with continuous audio. This is because the RNNs cannot be easily generalized to very long streams of audio when they are trained with segmented audio.

To address this problem, we propose an RNN training approach on training sequences with virtually infinite length. Specifically, we describe an efficient GPU-based RNN training framework for the truncated backpropagation through time (BPTT) algorithm, which is suitable for online (continuous) training. Then, we present an online version of the connectionist temporal classification (CTC) loss computation algorithm, where the original CTC loss is estimated with partial sliding window. This modified CTC algorithm can be directly employed for truncated BPTT based RNN training.

In addition, a fully RNN based end-to-end online ASR model is proposed. The model is composed of an acoustic RNN with CTC output and a character-level RNN language model that is augmented with a hierarchical structure. Prefix-tree based beam search decoding is employed with a new beam pruning algorithm to prevent exponential growth of the tree. The model is free from phoneme or lexicon models, and can be used for decoding infinitely long audio sequences. Also, this model has very small memory footprint compared to the other end-to-end systems while showing the competitive accuracy.

Furthermore, we propose an improved character-level RNN LM with a hierarchical structure. This character-level RNN LM shows improved perplexity compared to the lightweight word-level RNN LM with a comparable size. When this RNN LM is applied to the proposed character-level online ASR, better speech recognition accuracy can be achieved with reduced amount of computation.

Keywords : Automatic speech recognition (ASR), recurrent neural network (RNN), end-to-end learning, online inference

Student Number : 2010-23300

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Automatic Speech Recognition

Automatic speech recognition (ASR) [1] is a sequence to sequence mapping task, where the input is generally a sequence of acoustic feature vectors that are extracted from the given waveform, and the output is the corresponding transcription. Finding this mapping function is a regression problem, in which the model parameters of the function can be optimized using a speech corpus, which is a set of acoustic data and ground-truth transcriptions. However, the amount of text data in the transcriptions is usually not enough for the model to learn the linguistic structure. Therefore, in many cases, a large amount of additional text data is used to improve the language modeling performance.

1.1.1 Traditional ASR

Traditional ASR systems have usually employed a hidden Markov model (HMM) [2] to compute the probability of observing a given speech utterance for each possible transcription. In HMM-based systems, the probability of observing the current frame of the input sequence for a particular hidden state had been modeled with Gaussian mixture models (GMMs), until it was shown that feed-forward deep neural networks (DNNs) [3] can replace GMMs for better accuracy [4, 5]. Since the input of GMM or DNN is an acoustic feature over a short time window, the granularity of the hidden states needs to be equal to or smaller than phoneme-level. Therefore, a complete ASR system requires many levels of hierarchy such as phoneme, lexicon, and grammar networks. A weighted finite state transducer (WFST) [6] is often employed to represent each network, which allows all the networks to be easily combined into one with powerful composition and optimization algorithms. However, the resulting WFST is still very large and computing the probability of observing the given speech for every possible transcription is intractable in practice. Therefore, the Viterbi beam search is employed to efficiently find the best hypothesis of transcriptions with a reasonable amount of computation.

There are several drawbacks in the HMM-based model mainly due to its hierarchical structure. First, some of valuable information in speech (e.g. prosody) cannot contribute to the final recognition result, since it is not propagated to the higher level. Therefore, it is hard to achieve the human-level speech recognition performance with the traditional ASR system [7], considering that the human uses this information in many situations, such as for distinguishing homophones [8]. Second, designing and training this ASR system is complicated and requires a lot of effort for fine-tuning.

Moreover, there is no guarantee that the model structure optimized for a specific language will also work well on other languages. To remedy these issues, there have been many studies to remove these structural limitations of traditional ASR models as described in the next section.

1.1.2 End-to-End ASR with Recurrent Neural Networks

Unlike traditional HMM-based ASR models, it has been shown that recurrent neural networks (RNNs) [9] are capable of directly converting the input speech to the output text without any explicit phoneme or lexicon modeling [10, 11]. This is possible because RNNs have internal memory and can learn the long-term dependency between the input and output sequences. To be specific, RNNs can learn the direct mapping between the sequences of the input speech features and the output labels, where the granularity of the output labels can be characters or words. Since there are no intermediate layers between the speech and the text, these approaches are called "end-to-end". With a sufficient amount of training data, the end-to-end ASR systems have proven to show the human-level speech recognition accuracy on read speech [11].

1.1.3 Offline and Online ASR

In offline ASR, there is no constraint on the latency between the input speech and the output text. Therefore, the offline approach is applied to tasks where the accuracy is more important than the latency. On the other hand, the online ASR, or incremental speech recognition (ISR) [12], is more focused on the decoding latency, and usually employed for real-time applications such as spoken dialog systems or real-time automatic captioning. With traditional HMM-based ASR systems, the difference between

the offline and the online ASR is mostly in the decoding stage, and the related studies have been focused on the stability and accuracy of partial decoding results [13, 14].

However, the transition from offline to online systems is more complicated when it comes to RNN-based ASR. First, it is not trivial to use bidirectional RNNs [15] for online inference because of the latency problem. The bidirectional RNNs have a backward layer, where the signal is propagated from the future to the past. Since this backward signal is originated from the end of the sequence, it is impossible to start decoding until the input speech reaches to the end. Therefore, unidirectional RNNs are often employed for the online inference while sacrificing some recognition accuracy. Second, hidden states of RNNs are prone to explosion when the input sequence is very long [16]. This is not a problem for the offline ASR, where the input speech is pre-segmented. However, the online ASR is generally expected to be able to decode very long and arbitrary speech without any segmentation of input speech. In this case, the RNN explosion becomes a problem and should be prevented.

1.2 Scope of the Dissertation

1.2.1 End-to-End Online ASR with RNNs

As described in the previous sections, the online ASR is suitable for the applications where low-latency is required. However, applying end-to-end models for online ASR is not trivial due to the characteristics of RNNs. This dissertation tackles these issues and proposes end-to-end online ASR models that are based on RNNs.

The bidirectional RNNs are not covered in this dissertation, since they are not suitable for the online ASR due to the decoding latency. Applying the bidirectional architecture for the online ASR is another research area, where most of the contents

of this dissertation still apply [17]. Also, note that there has been effort to reduce the performance gap between the bidirectional and unidirectional RNNs by improving the architectures of the unidirectional ones [11].

The proposed ASR system in this dissertation consists of an acoustic model (AM) and a language model (LM), where both models are based on unidirectional RNNs. The AM is end-to-end trained with connectionist temporal classification (CTC) loss [18], and converts the input speech to characters. The LM is a character-level generative model [19], which predicts the probabilities of the next characters using the history of previous character inputs. By combining these two character-level models with prefix-tree based beam search, the complete ASR system can naturally handle out-of-vocabulary (OOV) words. Moreover, it is capable of decoding infinitely long speech with low latency. Also, the system demands a very small number of parameters compared to the WFST based ones while showing the comparable word error rate (WER).

1.2.2 Challenges and Contributions

Since training RNNs with modern computers generally takes very long time, it is important to build an efficient training system. However, unlike the stochastic gradient descent (SGD) [20] based training of feedforward DNNs [21], there is not much parallelism to utilize in the training algorithms for RNNs. In Chapter 2, a graphics processing unit (GPU) based flexible RNN training algorithm is presented. This algorithm analyzes a graph-based RNN architecture and automatically utilizes the parallelism. The training system described in this chapter is suitable for training very long sequences and used for training AMs and LMs throughout the dissertation.

As presented previously, RNNs have an explosion problem when decoding very

long sequences. One of the solutions to prevent this problem is also using long training sequences in the training time. This is very trivial problem for training RNN LMs when we use the training system presented in Chapter 2. However, to apply CTC loss for training AMs, the RNNs should be unrolled over the whole frames of the training sequence, which limits the length of the training sequences due to the memory capacity of modern GPUs. This problem is solved in Chapter 3 by modifying the CTC forward-backward algorithm so that the approximate gradients of the CTC loss can be computed within a sliding-window. Moreover, it is shown that the training time can be significantly reduced by employing a small sliding window and increasing the parallelism.

In Chapter 4, the character-level end-to-end online ASR system is described and analyzed. As described previously, this model is composed of an RNN-based AM and LM, and is capable of decoding infinitely long speech. Also, the model does not require lexicon and naturally handles OOV words. Since the beam search is performed on a prefix-tree, it is important to prevent the indefinite growth of the decoding tree, especially when the input speech is very long. In this chapter, *depth pruning* is introduced to solve this problem.

The major drawback of the model introduced in Chapter 4 is a large amount of computation required for inference of the character-level RNN LM. This problem is tackled in Chapter 5, where hierarchical RNN structures are proposed for character-level LMs. With the proposed hierarchical RNNs, it is shown that a better perplexity can be achieved with significantly less number of model parameters, which leads to improved accuracy and reduced amount of computations for the proposed ASR.

Note that large portions of the materials of Chapter 2, 3, 4, and 5 were previously published by the author in [22, 23, 24, 25], respectively.

Chapter 2

Flexible and Efficient RNN Training on GPUs

2.1 Introduction

Deep neural networks have shown quite impressive performances in several pattern recognition applications [3, 4]. Among the deep neural networks, the feed-forward networks are suitable for processing input data with a fixed length, and they are usually used for image and phoneme recognition. On the other hand, recurrent neural networks (RNNs) employ feedback inside, and they are suitable for processing input data whose dimension is not fixed. For example, automatic speech recognition (ASR) systems perform very well with an RNN-based language modeling [26].

Since RNNs contain feed-back loops inside, the past input can be memorized and affect the current output. If RNNs are properly trained, it is possible to compress the input history effectively and yield good results even when there are considerable time delays between the input and output. Especially, the long short-term memory

(LSTM) RNN is known to solve the problems with long time lag very successfully [27].

However, the LSTM RNN employs a very complex component known to be the memory block. It demands much effort even for slight modification of the structure because of the difficulty in deriving the corresponding training equation. Thus, it is needed to develop a generalized RNN structure that can be modified easily while representing LSTM networks perfectly. Previously, a generalized LSTM-like RNN structure with real-time recurrent learning (RTRL) [28] was proposed in [29] with special gated connections. However, we propose a much more general structure by introducing multiplicative layers and delayed connections. Also, we derive a back-propagation through time (BPTT) [30] based training algorithm for our RNN structure, which is generally more flexible than the RTRL-based one.

RNNs also demand very long training time, thus efficient implementation with GPUs or multiprocessors is needed. However, parallelization of the network is difficult due to dependency induced by the internal feedback loops. The conventional approach uses independent multiple training streams that employs plural copies of the network [31]. However, this inter-stream parallelism demands huge memory, which is a serious bottleneck for GPU based implementations.

In this chapter, we propose a parallelization approach as well as the generalized RNN structure. For this purpose, we first develop training algorithms for the generalized RNNs. The training equations of conventional LSTM can be perfectly represented with the generalized equations. Then, the parallelization approach exposes single-stream parallelization (intra-stream parallelism) that does not increase the size of mini-batches as the conventional multi-stream parallelization (inter-stream parallelism). Experimental results show that further speed-up can be achieved by combin-

ing both intra-stream and inter-stream parallelization.

This chapter is organized as follows. The generalized LSTM-like RNN structure is proposed and its training equations are derived in Section 2.2. In Section 2.3, the intra-stream parallelism of the generalized RNNs is explored and combined with the conventional inter-stream parallelism. In Section 2.4, experimental results of the proposed approach on a GPU are presented, followed by concluding remarks in Section 2.5.

2.2 Generalization

To apply our parallelization approach to various types of RNNs, we first introduce a generalized RNN structure that can represent complex RNNs using simple basic blocks. This generalization fully covers advanced LSTM network structures with forget gates and peephole connections, and their BPTT-based training algorithm. Also, with the generalized RNN, one can easily design a new RNN structure quite easily since the corresponding equations and the parallelization approach remain the same.

2.2.1 Generalized RNN Structure

The proposed generalized RNN structure is basically a directed graph, which consists of a set of nodes and edges. Each node represents a layer and each edge makes a connection between two layers. There are two types of connections: delayed or not. A delayed connection makes a fixed amount of delay on the signal, and is used to construct a recurrent loop. More specifically, the connection m propagates the activation

of the source layer k at the frame $t - d_m$ to the destination layer at the frame t as

$$\mathbf{z}_m(t) = \mathbf{W}_m \mathbf{y}_k(t - d_m), \tag{2.1}$$

where $\mathbf{z}_m$ is the output of the connection m, W_m is the corresponding weight matrix, $\mathbf{y}_k$ is the activation of the source layer k, and d_m is the amount of delay at the connection m. The value of d_m is 0 for non-delayed connections and larger than 0 for delayed connections.

In an additive layer, the inputs are summed up and the activation function is applied on it:

$$\mathbf{s}_k(t) = \sum_{m \in A_k} \mathbf{z}_m(t) \tag{2.2}$$

$$\mathbf{y}_k(t) = f_k(\mathbf{s}_k(t)), \tag{2.3}$$

where $\mathbf{s}_k$ is the state (input), A_k is the set of the indices of the anterior connections, $\mathbf{y}_k$ is the activation, and $f_k(\cdot)$ is the activation function of the layer k. In addition to the normal additive layers, multiplicative layers are employed to represent gate units of LSTM networks. A multiplicative layer performs element-wise multiplication of input vectors (or matrices for batched computation) as follows:

$$s_{k,i}(t) = \prod_{m \in A_k} z_{m,i}(t), \tag{2.4}$$

where the subscript i represents the index of elements in a vector.

For generality, we introduce an aggregation function $g_k(\cdot)$ as

$$\mathbf{s}_k(t) = g_k(\{\mathbf{z}_m(t) | m \in A_k\}), \tag{2.5}$$

where $g_k(\cdot)$ is a vector addition function for an additive layer or an element-wise multiplication function for a multiplicative layer, or it can be other nonlinear functions to add further nonlinearity to the network.

In the previous approach on the generalized LSTMs [29], the gate units are implemented with gated connections. However, the gated connection has two input layers, so it cannot be regarded as an edge of the standard directed graph structure, where each edge has one input and one output.

In our approach, by introducing the multiplicative layers, LSTM gates can be regarded as normal nodes in a graph structure, which allows general graph algorithms to be directly applied in Section 2.3. As an example, Figure 2.1 shows a generalized representation of a single-layer LSTM network with forget gates and peephole connections.

2.2.2 Training

In this section, BPTT [30] based training equations for the generalized RNN are derived. The objective is to minimize the following total error from $t_0 + 1$ to t_1:

$$E^{total}(t_0, t_1) = \sum_{t_0 < t \leq t_1} E(t), \tag{2.6}$$

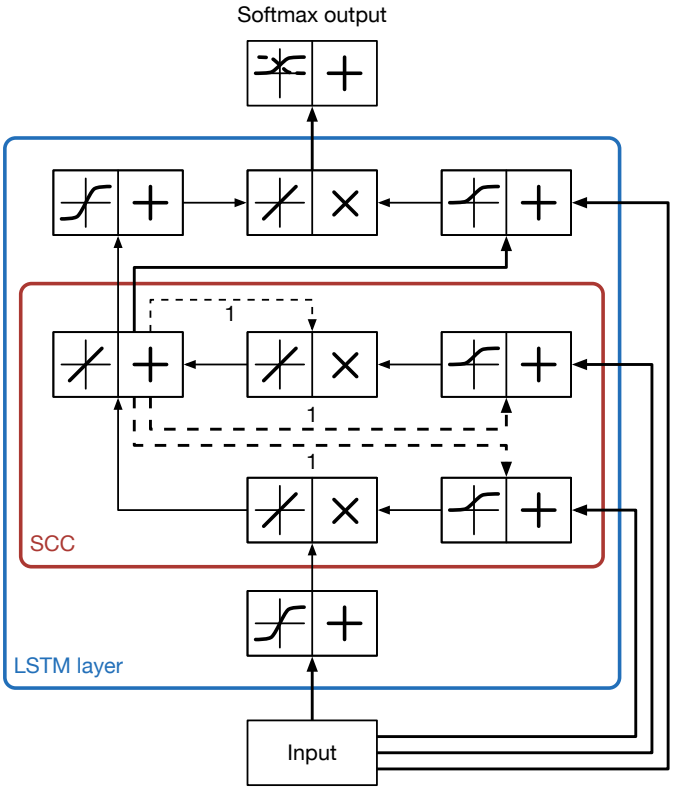


Figure 2.1: Generalized representation of an LSTM network with forget gates and peephole connections. Thick arrows represent connections with full weight matrices. On the other hand, connections with thin arrows have identity weight matrices. The numbers on the dashed lines indicate the corresponding delay amounts. A non-singleton strongly connected component (SCC) is drawn, of which nodes will be grouped into a single recurrent node to make the network acyclic.

where $E(t)$ is the error at frame t. For convenience, we define two derivative variables as

$$\delta_{k,i}(t) = -\frac{\partial E^{total}(t_0,t_1)}{\partial s_{k,i}(t)} \tag{2.7}$$

$$\varepsilon_{m,i}(t) = -\frac{\partial E^{total}(t_0,t_1)}{\partial z_{m,i}(t)}. \tag{2.8}$$

These two variables will be back-propagated at the backward pass. If the layer k is an output layer, $\delta_{k,j}(t)$ should be initialized by comparing the output with a desired output $d_{k,j}(t)$ according to the error criterion defined by $E(t)$ and the activation function of the output layer. Using the minimum cross-entropy criterion with the softmax activation function,

$$\delta_{k,j}(t) = d_{k,j}(t) - y_{k,j}(t). \tag{2.9}$$

If the layer k is not an output layer,

$$\delta_{k,j}(t) = -\sum_{n\in P_k}\sum_{i\in I_n} \frac{\partial E^{total}(t_0,t_1)}{\partial z_{n,i}(t+d_n)} \frac{\partial z_{n,i}(t+d_n)}{\partial y_{k,j}(t)} \frac{\partial y_{k,j}(t)}{\partial s_{k,j}(t)} \tag{2.10}$$

$$= \sum_{n\in P_k}\sum_{i\in I_n} \varepsilon_{n,i}(t+d_n) W_{n,ij} f_k'(s_{k,j}(t)), \tag{2.11}$$

where P_k is the set of posterior connection indices of the layer k and I_n is the set of element indices of the vector z_n. Also, $\varepsilon_{m,j}(t)$ becomes

$$\varepsilon_{m,j}(t) = -\frac{\partial E^{total}(t_0,t_1)}{\partial s_{k,j}(t)} \frac{\partial s_{k,j}(t)}{\partial z_{m,j}(t)} \tag{2.12}$$

$$= \delta_{k,j}(t) \frac{\partial}{\partial z_{m,j}(t)} g_k(\{\mathbf{z}_n(t) | n \in A_k\}), \tag{2.13}$$

where k is the index of the destination layer of the connection m. To truncate errors at $t = t_0'$, we backpropagate the two derivative variables while $t > t_0'$ where $t_0' \leq t_0$ using (2.11) and (2.13). After the backward pass, the truncated error gradient of the connection $m \in P_k$ can be acquired by

$$\frac{\partial E^{total}(t_0,t_1)}{\partial W_{m,ij}} \approx \sum_{t_0'<t\leq t_1} \frac{\partial E^{total}(t_0,t_1)}{\partial z_{m,i}(t)} \frac{\partial z_{m,i}(t)}{\partial W_{m,ij}} \tag{2.14}$$

$$= - \sum_{t_0'<t\leq t_1} \varepsilon_{m,i}(t) y_{k,j}(t-d_m). \tag{2.15}$$

In matrix form, (2.11) can be represented as

$$\delta_k(t) = \left(\sum_{n\in P_k} \mathbf{W}_n^T \varepsilon_n(t+d_n) \right) \circ f_k'(\mathbf{s}_k(t)), \tag{2.16}$$

where $\circ$ denotes element-wise vector multiplication. If the layer k is an additive layer, then (2.13) becomes

$$\varepsilon_m(t) = \delta_k(t). \tag{2.17}$$

Otherwise for the multiplicative layer k,

$$\varepsilon_m(t) = \delta_k(t) \circ \overset{\circ}{\prod_{n\in A_k, n\neq m}} \mathbf{z}_n(t), \tag{2.18}$$

where element-wise multiplications are performed with $\prod$. The error gradient matrix for the connection $m \in P_k$ is computed by

$$\nabla \mathbf{W}_m = - \sum_{t_0'<t\leq t_1} \varepsilon_m(t) \mathbf{y}_k^T(t-d_m). \tag{2.19}$$

The error gradients can be used for the first order optimization methods such as stochastic gradient descent.

2.3 Parallelization

Parallelization of RNN computation is quite challenging due to dependencies between two consecutive frames. The state of an RNN of the frame k cannot be determined until the computation for the frame $k-1$ is finished. In this section, we first develop a parallelization method for the forward and the backward pass with a single stream (intra-stream parallelism), and then extend the approach to a multi-stream case (inter-stream parallelism).

2.3.1 Intra-Stream Parallelism

The key concept of separating sequential parts from the parallel parts of an RNN is to determine loops in the RNN and group each loop into a single special node called a *recurrent node*. Then, the remaining structure becomes a directed acyclic graph (DAG), which can be easily parallelized as in a mini-batch based feed-forward neural network computation. Only the internal computations of the recurrent nodes are performed sequentially.

More specifically, strongly connected components (SCCs) are found to determine which nodes should be grouped into a recurrent node. An SCC is a subgraph that is strongly connected, that is, there are one or more paths between every pair of two vertices inside the subgraph. An SCC analysis finds a set of SCCs that form a partition of the vertex set of the original graph. For SCCs that are singletons and do not contain a self-loop, the original nodes inside the SCCs remain unchanged. Otherwise, the

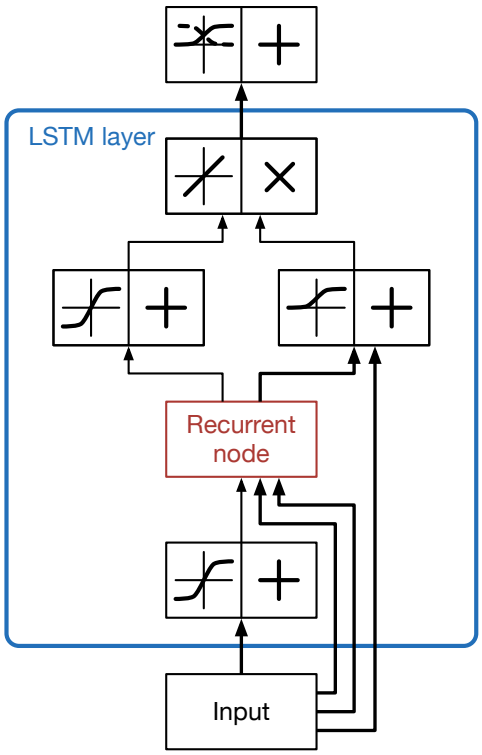


Figure 2.2: Feed-forward representation of the LSTM network that is depicted in Figure 2.1.

nodes in each SCC are grouped into a single recurrent node. Then, the final graph becomes a DAG and be ready for parallel computation. An example of an LSTM network is shown in Figure 2.2. One of the famous algorithms for finding SCCs is the Tarjan's strongly connected component algorithm [32]. Tarjan's algorithm also provides a reverse topological sort of the resulting DAG, which is useful to determine the activation order.

Once an RNN is represented as a DAG, the forward computation becomes very similar to that of feedforward networks. As in the case of feedforward networks, computations of nodes and edges are performed in a topological order of the DAG.

These operations can be done in parallel over several frames since the network is represented as a DAG and there are no dependencies between different frames except the isolated recurrent nodes.

Recurrent nodes are subgraphs of the original RNN and should be computed sequentially. The computation of a recurrent node from frame t_0 to t_1 in the forward pass requires $t_1 - t_0 + 1$ sequential steps. In each step of the forward pass, delayed connections are computed first. Then the remaining part excluding the delayed connections becomes a DAG and can be computed in a topological order. The computation of a backward pass can be performed similarly with reversed topological orders.

The sequential computations of recurrent nodes are quite expensive and often become a bottleneck of the overall performance. To speed up these sequential parts, we need to employ the multi-stream parallelization.

2.3.2 Inter-Stream Parallelism

Inter-stream parallelism can be explored in the multi-stream mode where an RNN processes N streams with independent contexts. This is equivalent to running N independent copies of the RNN. Therefore, the multi-stream mode greatly increases parallelism and the overall execution speed. Recently, this approach was successfully applied to speed up language model training with an Elman network on a GPU [31].

For training an RNN in the multi-stream mode, the input and target streams are usually given by connecting randomly ordered training sequences. Since the lengths of the training sequences are very long, we apply the efficient version of truncated BPTT(h), denoted as BPTT($h;h'$) proposed in [33]. BPTT($h;h'$) is similar to the ordinary truncated BPTT(h) in that the network is unrolled h times. However, in the forward pass of BPTT($h;h'$), h' time steps are computed at once. Also, the error

gradients for the recent h' output errors are obtained by one iteration. These error gradients are summed up over the N training streams. Therefore, output errors of total $N \times h'$ frames affect the error gradients when updating weights after backward passes. We call the set of these frames as a *mini-batch* throughout the chapter, as it is equivalent to a mini-batch in stochastic gradient descent methods of feedforward neural networks.

Increasing N also speeds up the training. However, we cannot make N very large since the size of a mini-batch, $N \times h'$, is limited by the physical memory size of a GPU. Moreover, increasing the size of a mini-batch results in infrequent update of the weights and may slow down the convergence [34]. Also, the parameter h' cannot be easily modified since the training speed is approximately proportional to the ratio of h' to h. For simplicity, let us assume $h = 2h'$ to fix the training speed. In this case, error propagates through h' to $2h' - 1$ previous time steps in the backward pass. Therefore h' should be set sufficiently large to solve long time lag problems.

2.4 Experiments

Nvidia Tesla K40 GPU is used for the following experiments. For all experiments, BPTT($2h$; h) is used for simplicity. Since the training algorithm for the generalized RNN structure is mathematically equivalent to that of Elman or LSTM networks, the results regarding performance measures such as the accuracy or the perplexity are not reported.

To evaluate the proposed parallelization approach, we evaluate the language model training speed with the multi-stream mode as in [31]. The RNN architecture is an Elman network with 38,000 input, 512 hidden, and 20,000 output units. The mini-

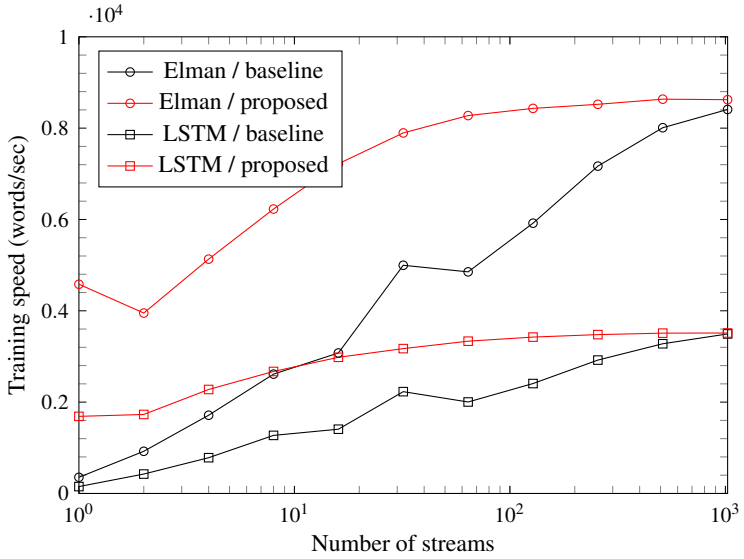


Figure 2.3: Comparison of language model training speeds with Elman and LSTM networks. The LSTM employs forget gates and peephole connections. The sizes of the input layer, hidden or LSTM layer, and output layer is 38,000, 512, and 20,000 respectively. The mini-batch size is fixed to 1,024, so the error propagates from $1{,}024/N$ to $2{,}048/N-1$ previous steps where N is the number of streams.

batch size is fixed to 1,024 to use the same amount of GPU memory. Hence, with N streams, $h = 1{,}024/N$ and the error propagates from $1{,}024/N$ to $2{,}048/N-1$ previous time steps. For comparison, an LSTM version of the network with forget gates and peephole connections are also evaluated. Note that the LSTM network has no self recurrent connection from the output of the LSTM layer to the input of that.

The training speeds are compared in Figure 2.3 with varying number of streams. Since the baseline approaches does not exploit intra-stream parallelism, they show poor training speeds when the number of streams are small. On the other hand, the proposed approach employs intra-stream parallelism and shows over 10 times of speed-up over the baseline approach when a single stream is used. Also, with the proposed approach, we can obtain almost the maximum speed only with 64 streams.

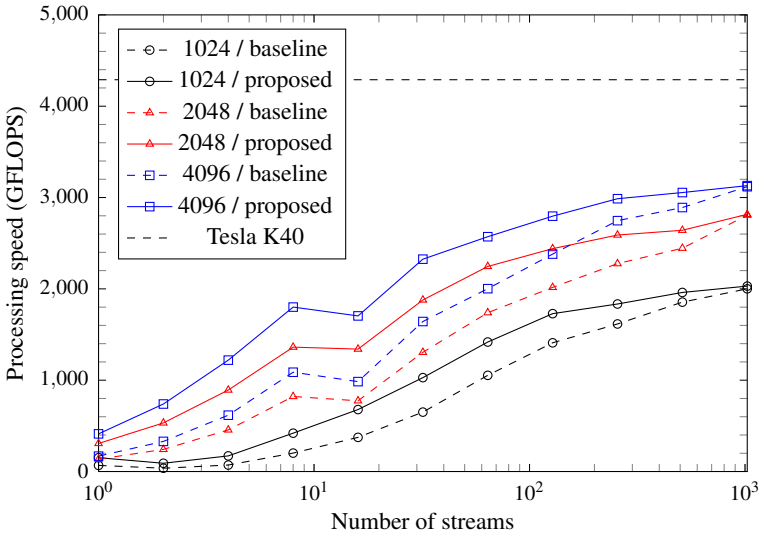


Figure 2.4: Comparison of GPU processing power utilizations when training LSTM networks with the three different sizes of LSTM layers: 1,024, 2,048, and 4,096. The input and output layers have the same size as the LSTM layer. Also, the theoretical peak performance of Tesla K40 GPU is shown. The mini-batch size is fixed to 1,024.

This is a nice advantage since using less number of streams allows RNNs to learn longer time lags when the size of mini-batch is limited, as discussed in Section 2.3.2.

To analyze scalability and GPU efficiency with various size of networks, we perform another experiment with LSTM networks with forget gates and peephole connections. All layers of each network have the same size, which is 1,024, 2,048, or 4,096. To examine the GPU utilizations, we present the number of single-precision floating point operations per second (FLOPS) in Figure 2.4 along with the theoretical peak performance of Tesla K40 GPU. Note that only the operations for parameters and error gradients are counted. Compared to the previous experiment where the input and output layers are very large, this example is much closer to the deep RNN architectures in terms of the ratio of the sequential computations (inside the recurrent

nodes) to the parallel computations. As shown in the figure, the GPU utilization gets higher as the layer size or the number of streams increases. Also, the intra-stream parallelism further accelerates the training especially with the small number of streams.

2.5 Concluding Remarks

We introduced a generalized structure for RNNs which covers LSTM networks with forget gates and peephole connections. This generalized structure is represented as a directed graph where nodes and edges correspond to layers and connections, respectively. Due to the graph representation, we can automatically find loops inside RNNs using the Tarjan's strongly connected component algorithm and explore intra-stream parallelism. The proposed intra-stream parallelism is combined with inter-stream parallelism in the multi-stream mode training for further acceleration. The experiments show that exploiting these two parallelisms greatly speeds up the training task on a GPU.

Chapter 3

Online Sequence Training with Connectionist Temporal Classification

3.1 Introduction

Supervised sequence learning is a regression task where the objective is to learn a mapping function from the input sequence $\mathbf{x}$ to the corresponding output sequence $\mathbf{z}$ for all $(\mathbf{x}, \mathbf{z}) \in S$ with the given training set S, where $\mathbf{x}$ and $\mathbf{z}$ can have different lengths. When combined with recurrent neural networks (RNNs), supervised sequence learning has shown great success in many applications including machine translation [35, 36, 37], speech recognition [38, 39, 10, 40, 41, 42, 43, 44, 45], and handwritten character recognition [46, 47]. Although several attention-based models have been introduced recently, connectionist temporal classification (CTC) [18] is still one of the most successful techniques in practice, especially for end-to-end speech and character recognition tasks [10, 40, 44, 45, 46, 47].

The CTC based sequence training is usually applied to bidirectional RNNs [15],

where both the past and the future information is considered for generating the output at each frame. However, the output of the bidirectional RNNs is available after all of the frames in the input sequence are fed into the RNNs because the future information is backward propagated from the end of the sequence. Therefore, the bidirectional RNNs cannot be employed for low-latency online applications such as incremental speech recognition (ISR) [48], especially for the character-level ISR system, which will be proposed in Chapter 4. On the other hand, unidirectional RNNs only make use of the past information and are suitable for low-latency applications at the cost of a little accuracy loss. Moreover, the CTC-trained unidirectional RNNs do not need to be unrolled (or unfolded) at the test time. It is shown by [49] that CTC can also be employed for sequence training of unidirectional RNNs on a phoneme recognition task. In this case, the unidirectional RNN also learns the suitable amount of the output delay that is required to accurately process the input sequence. The work in [50] reports that when a CTC-trained unidirectional RNN is employed for online spoken term detection, the detection latency becomes around 200 ms, which is similar to human response time to speech stimuli [51].

For the CTC training of both unidirectional and bidirectional RNNs, it is required to unroll the RNNs by the length of the input sequence. By unrolling an RNN N times, every activations of the neurons inside the network are replicated N times, which consumes a huge amount of memory especially when the sequence is very long. This hinders a small footprint implementation of online learning or adaptation. Also, this "full unrolling" makes a parallel training with multiple sequences inefficient on shared memory models such as graphics processing units (GPUs), since the length of training sequences is usually not uniform, and thus a load imbalance problem occurs. This load imbalance problem can be solved by grouping training se-

quences with similar lengths into buckets [43, 36]. However, it is difficult to achieve high parallelism with this approach, when the training sequences are very long. For unidirectional RNNs, this problem can be addressed by concatenating sequences to make a very long stream of sequences, and training the RNNs with synchronized fixed-length unroll-windows over multiple training streams, as described in Chapter 2. However, it is not straightforward to apply this approach to the CTC training, since the standard CTC algorithm requires full unrolling for the backward variable propagation, which starts from the end of the sequence.

In this chapter, we propose an expectation-maximization (EM) based online CTC algorithm for sequence training of unidirectional RNNs. The algorithm allows training sequences to be longer than the amount of the network unroll. Moreover, it can be applied to infinitely long input streams with roughly segmented target sequences (e.g. only with the utterance boundaries and the corresponding transcriptions for training an end-to-end speech recognition RNN). It was shown that the resulting RNN can run continuously without pre-segmentation or external reset, and is useful for the continuous spoken term detection [50] and low-latency ISR systems, which will be described in Chapter 4 in detail, where the input speech is infinitely long. Due to the fixed unroll amount, we expect that the proposed algorithm is suitable for online semi-supervised learning or adaptation systems with constrained hardware resource. Furthermore, the approach can directly be combined with the GPU based parallel RNN training algorithm described in Chapter 2. For evaluation, we present examples of end-to-end speech recognition on the Wall Street Journal (WSJ) corpus [52] with continuously running RNNs. Also, further experiments are performed on TIMIT [53] and the results are compared with others in literature. Experimental results show that the proposed online CTC algorithm performs comparable to the almost fully un-

rolled CTC training even with the small unroll amount that is shorter than the average length of the sequences in the training set. Also, the reduced amount of unroll allows more parallel sequences to be trained concurrently with the same memory use, which results in greatly improved training speed on a GPU.

The chapter is organized as follows. In Section 3.2, the standard CTC algorithm is explained. Section 3.3 contains the definition of the online sequence training problem and proposes the online CTC algorithm. In Section 3.4, the algorithm is extended for the continuously running RNNs, which is followed by its parallelization in Section 3.5. In Section 3.6, the proposed algorithm is evaluated with speech recognition examples. Concluding remarks follow in Section 3.7.

3.2 Connectionist Temporal Classification

The CTC algorithm [18, 49] uses a many-to-one sequence-to-sequence mapping function that converts the sequence of labeling with timing information (e.g. frame-wise output labels from RNNs) into the shorter sequence of labels by removing timing and alignment information. The main idea is to introduce the additional CTC blank label, b, for the sequence that has timing information, and remove the blank labels and merging repeating labels to obtain the unique corresponding sequence.

Specifically, for the set of target labels, L, and its extended set with the additional CTC blank label, $L' = L \cup \{b\}$, the path, π, is defined as a sequence over L', that is, $\pi \in L'^T$, where T is the length of the input sequence, $\mathbf{x}$. Then, the output sequence, $\mathbf{z} \in L^{\leq T}$, is represented by $\mathbf{z} = \mathscr{F}(\pi)$ with the sequence to sequence mapping function $\mathscr{F}$. $\mathscr{F}$ maps any path π with the length T into the shorter sequence of the label, $\mathbf{z}$, by first merging the consecutive same labels into one and then removing the blank

labels. Therefore, any sequence of the raw RNN outputs with the length T can be decoded into the shorter labeling sequence, $\mathbf{z}$, with ignoring timings. This enables the RNNs to learn the sequence mapping, $\mathbf{z} = \mathscr{G}(\mathbf{x})$, where $\mathbf{x}$ is the input sequence and $\mathbf{z}$ is the corresponding target labeling for all $(\mathbf{x}, \mathbf{z})$ in the training set, S. More specifically, the gradient of the loss function $\mathscr{L}(\mathbf{x}, \mathbf{z}) = -\ln p(\mathbf{z}|\mathbf{x})$ is computed and fed to the RNN through the softmax layer [54], of which the size is $|L'|$.

As depicted in Figure 3.1, the CTC algorithm employs the forward-backward algorithm for computing the gradient of the loss function, $\mathscr{L}(\mathbf{x}, \mathbf{z})$. Let $\mathbf{z}'$ be the sequence over L' with the length of $2|\mathbf{z}| + 1$, where $z'_u = b$ for odd u and $z'_u = z_{u/2}$ for even u. Then, the forward variable, α, and the backward variable, β, are initialized by

$$\alpha(1,u) = \begin{cases} y^1_b & \text{if } u = 1 \\ y^1_{z_1} & \text{if } u = 2 \\ 0 & \text{otherwise} \end{cases} \qquad \beta(T,u) = \begin{cases} 1 & \text{if } u = |\mathbf{z}'|, |\mathbf{z}'| - 1 \\ 0 & \text{otherwise} \end{cases}, \tag{3.1}$$

where y^t_k is the softmax output of the label $k \in L'$ at time t.

The variables are forward and backward propagated as

$$\begin{aligned} \alpha(t,u) &= y^t_{z'_u} \sum_{i=f(u)}^{u} \alpha(t-1,i) \\ \beta(t,u) &= \sum_{i=u}^{g(u)} \beta(t+1,i) y^{t+i}_{z'_i}, \end{aligned} \tag{3.2}$$

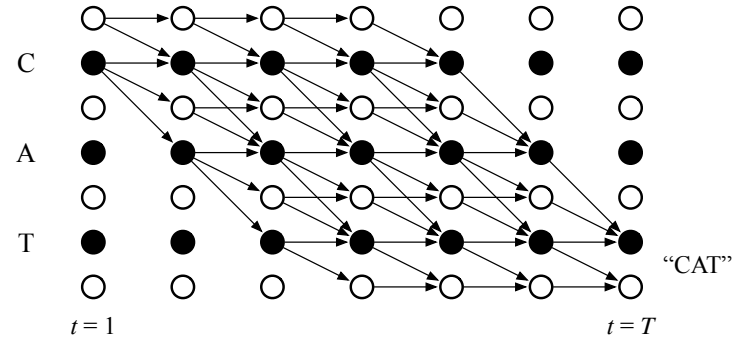


Figure 3.1: CTC forward-backward example for the target sequence "CAT", where the black and white dots represent the labels and blanks, respectively. The arrows indicate the allowed transitions.

where

$$f(u) = \begin{cases} u-1 & \text{if } z'_u = b \text{ or } z'_{u-2} = z'_u \\ u-2 & \text{otherwise} \end{cases}$$
$$g(u) = \begin{cases} u+1 & \text{if } z'_u = b \text{ or } z'_{u+2} = z'_u \\ u+2 & \text{otherwise} \end{cases} \tag{3.3}$$

with the boundary conditions:

$$\alpha(t,0) = 0,\ \forall t\ , \quad \beta(t,|\mathbf{z}'|+1) = 0,\ \forall t. \tag{3.4}$$

Then, the error gradient with respect to the input of the softmax layer at time t, a_k^t, is computed as

$$\frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial a_k^t} = y_k^t - \frac{1}{p(\mathbf{z}|\mathbf{x})} \sum_{u \in B(\mathbf{z},k)} \alpha(t,u)\beta(t,u), \tag{3.5}$$

where $B(\mathbf{z},k) = \{u : \mathbf{z}'_u = k\}$ and $p(\mathbf{z}|\mathbf{x}) = \alpha(T,|\mathbf{z}'|) + \alpha(T,|\mathbf{z}'|-1)$.

3.3 Online Sequence Training

3.3.1 Problem Definition

Throughout the chapter, the online sequence training problem is defined as follows.

- The training set S consists of pairs of the input sequence $\mathbf{x}$ and the corresponding target sequence $\mathbf{z}$, that is, $(\mathbf{x},\mathbf{z}) \in S$.
- The estimation model $\mathscr{M}$ learns the general mapping $\mathbf{z} = \mathscr{G}(\mathbf{x})$, where the training sequences $(\mathbf{x},\mathbf{z}) \in S$ are sequentially given.
- For each $(\mathbf{x},\mathbf{z}) \in S$ and at time t, only the fraction of the input sequence up to time t, $\mathbf{x}_{1:t}$, and the entire target sequence, $\mathbf{z}$, are given, where $1 \leq t \leq |\mathbf{x}|$. The length of the input sequence, $|\mathbf{x}|$, is unknown except when $t = |\mathbf{x}|$.
- The parameters of the estimation model $\mathscr{M}$ are updated in the manner of online learning, that is, they can be frequently updated even before seeing the entire input sequence $\mathbf{x}$.

This online learning problem usually occurs in real world when a human learns a language from texts and the corresponding audio. For example, when watching movies with subtitles, we are given the entire target sequence (subtitle for the current utterance) and the input sequence (the corresponding audio) up to the current time, t. We cannot access the future audio and even do not know exactly when the utterance will end (at $t = |\mathbf{x}|$).

Algorithm 1 Online CTC training with BPTT(h; h') for a single sequence

$\tau_0 \leftarrow 0$
$n \leftarrow 1$
while $\tau_{n-1} < T$ **do**
  $\tau'_n \leftarrow \max\{1, nh' - h + 1\}$
  $\tau_n \leftarrow \min\{nh', T\}$
  RNN forward activation from $t = \tau_{n-1} + 1$ to τ_n
  CTC(h; h') error computation on the output layer
  RNN backward error propagation from $t = \tau_n$ to τ'_n
  RNN gradient computation and weight update
  $n \leftarrow n + 1$
end while

When RNNs are trained with the standard CTC algorithm, it is difficult to determine how much amount of unrolling is needed before the entire sequence $\mathbf{x}$ is given, since the length of $\mathbf{x}$ is unknown at time $t < |\mathbf{x}|$. Also, it is not easy to learn the sequences that are longer than the unroll amount, which is often constrained by the hardware resources.

3.3.2 Overview of the Proposed Approach

We propose an online CTC algorithm where the RNN can learn the sequences longer than the unroll amount, h. The algorithm is based on the truncated backpropagation through time (BPTT) algorithm [30] with the forward step size of h' and the unroll amount of h, which is called BPTT(h; h'), as proposed in [33]. Algorithm 1 describes the BPTT(h; h') algorithm combined with the CTC loss, where T is the length of the training sequence, $\mathbf{x}$.

However, although BPTT(h; h') is designed for online training of RNNs, employing the standard CTC loss function requires full unrolling of the networks. Therefore, we propose the CTC(h; h') algorithm for computing the CTC loss in the online man-

Algorithm 2 CTC(h; h') at the iteration n

$\tau_{n-1} \leftarrow (n-1)h'$
$\tau'_n \leftarrow \max\{1, nh' - h + 1\}$
$\tau'_{n+1} \leftarrow \max\{1, (n+1)h' - h + 1\}$
$\tau_n \leftarrow \min\{nh', T\}$
if $n = 1$ **then**
    Init. CTC forward variable, α, at $t = 1$
end if
CTC forward prop. of α from $t = \tau_{n-1} + 1$ to τ_n
if $\tau_n = T$ **then**
    Init. CTC-TR backward variable, β, at $t = T$
    CTC-TR backward prop. of β from $t = T$ to τ'_n
    CTC-TR error computation on $t \in [\tau'_n, T]$
else
    Init. CTC-EM backward variable, β_{τ_n}, at $t = \tau_n$
    CTC-EM backward prop. of β_{τ_n} from $t = \tau_n$ to τ'_n
    CTC-EM error computation on $t \in [\tau'_n, \tau'_{n+1} - 1]$
    Set error to zero on $t \in [\tau'_{n+1}, \tau_n]$
end if

ner as in BPTT(h; h') as in Algorithm 2. The algorithm is also depicted in Figure 3.2 with the example in which the length of the sequence, $T = |\mathbf{x}|$, is 2.5 times as long as the unroll amount.

CTC(h; h') consists of two CTC algorithms. The first one is the truncated CTC (CTC-TR), which is basically the standard CTC algorithm applied at the last iteration with truncation. In the other iterations, the generalized EM based CTC algorithm (CTC-EM) is employed from $t = \max\{1, nh' - h + 1\}$ to $\max\{0, (n+1)h' - h\}$ with the modified backward variable, β_τ. The CTC-TR and CTC-EM algorithms are explained in Section 3.3.3 and Section 3.3.4, respectively. Note that simply setting $h = 2h'$ works well in practice. In this setting, we denote the algorithm as CTC($2h'$; h').

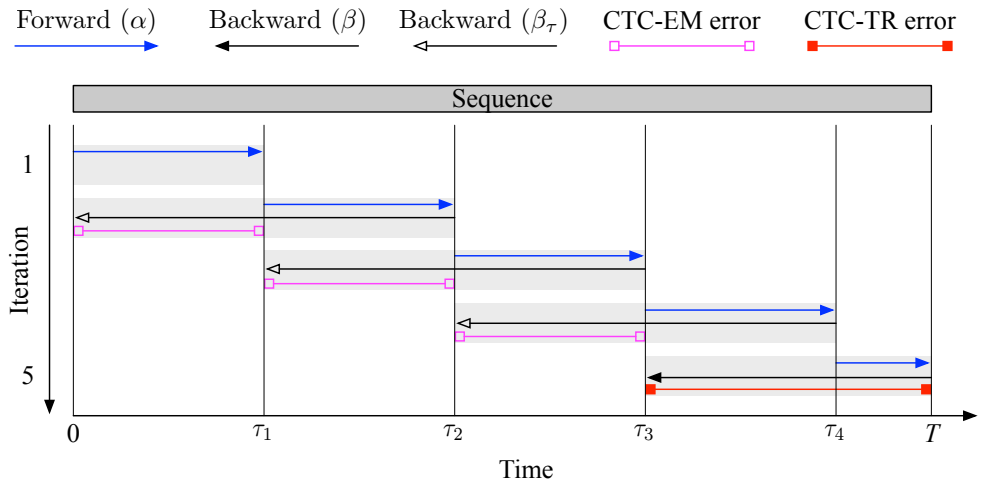


Figure 3.2: Online CTC($2h'$; h') algorithm depicted for a single sequence that is longer than the RNN unroll amount. The shaded areas indicate the range of the RNN unrolling at each iteration.

3.3.3 CTC-TR: Standard CTC with Truncation

With the standard CTC algorithm, it is not possible to compute the backward variables when $\tau_n < T$, as the future information beyond τ_n cannot be accessed. Therefore, we only compute the CTC errors at the last iteration, where $\tau_n = T$ as in Algorithm 2. In this case, however, the gradients are only available in the unroll range. Since the backward propagation is truncated at the beginning of the unroll range, we call the CTC algorithm in this range as truncated CTC, or CTC-TR. Also, we call the range that is covered by the CTC-TR algorithm as the CTC-TR coverage.

The RNN can be trained only with CTC-TR if there are sufficient labels that occur within the CTC-TR coverage. However, the CTC-TR coverage decreases by making the unroll amount smaller. Then, the percentage of the effective training frames, which actually generate the output errors, goes down, and the efficiency of train-

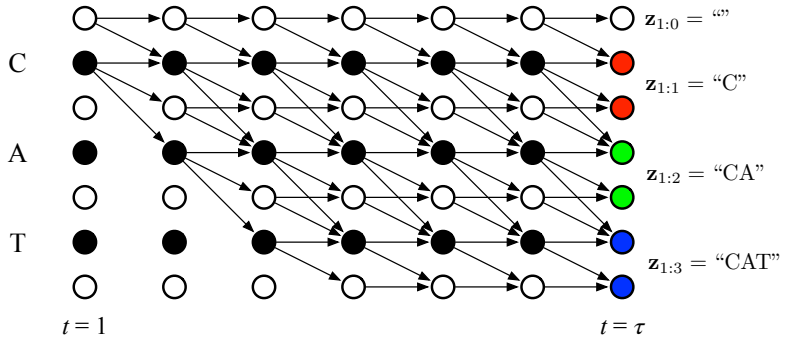


Figure 3.3: Forward-backward algorithm of CTC-EM for the target sequence “CAT”, where the black and white dots represent the labels and CTC blanks, respectively. The arrows represent the paths with allowed transitions.

ing decreases. Also, the effective size of the training set gets smaller, which results in the generalization performance loss of the RNN. Therefore, for maintaining the training performance while reducing the unroll amount, it is critical to make use of the full training frames by employing the CTC-EM algorithm, which is described in Section 3.3.4.

3.3.4 CTC-EM: EM-Based Online CTC

Assume that only the fraction of the input sequence, $\mathbf{x}_{1:\tau}$, is available. Then, as shown in Figure 3.3, there are $|\mathbf{z}|+1$ possible partial labelings.[1] Let $\mathbf{z}_{1:m}$ be the subsequence of $\mathbf{z}$ with the first m labels. Also we define Z as the set that consists of these labeling sequences:

$$Z = \{\mathbf{z}_{1:m} : 0 \leq m \leq |\mathbf{z}|\}. \tag{3.6}$$

[1] Although $\mathbf{z}_{1:m}$ is not possible by the standard CTC formulation when $m > \tau$, we can still say that $\mathbf{z}_{1:m}$ is a possible labeling with a probability of zero without loss of generality.

One of the most simple approach for training the network under this condition is to choose the most likely partial alignment from Z and compute the standard CTC error by regarding the partial alignment as the ground truth labeling. For example, we can select $\mathbf{z}_{1:m'}$ where $m' = \arg\max_m \alpha(\tau, m)$ since $\alpha(\tau, m)$ is a posterior probability $p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau}, \mathbf{w}^{(n)})$ with the current network parameter $\mathbf{w}^{(n)}$. This is a well-known hard-EM approach. This simple idea can easily be extended to the more sophisticated soft-EM approach as follows. First, select one of the partial labelings in Z with the probability $p(\mathbf{z}_{1:m}|Z, \mathbf{x}_{1:\tau}, \mathbf{w}^{(n)})$ estimated by the RNN with current parameters (E-step). Then, maximize the probability of that labeling by adjusting the parameters (M-step).

This optimization problem is readily reduced into the generalized EM algorithm. Specifically, the expectation step is represented as

$$\begin{aligned} Q_\tau(\mathbf{w}|\mathbf{x}, \mathbf{z}, \mathbf{w}^{(n)}) &= \mathbb{E}_{\mathbf{z}_{1:m}|Z, \mathbf{x}_{1:\tau}, \mathbf{w}^{(n)}} \left[\ln p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau}, \mathbf{w})\right] \\ &= \sum_{m=0}^{|\mathbf{z}|} p(\mathbf{z}_{1:m}|Z, \mathbf{x}_{1:\tau}, \mathbf{w}^{(n)}) \ln p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau}, \mathbf{w}), \end{aligned} \tag{3.7}$$

where $\mathbf{w}^{(n)}$ is the set of network parameters at the current iteration, n. In the maximization step of the generalized EM approach, we try to maximize Q_τ by finding new parameters $\mathbf{w}^{(n+1)}$ that satisfies $Q_\tau(\mathbf{w}^{(n+1)}|\mathbf{x}, \mathbf{z}, \mathbf{w}^{(n)}) \geq Q_\tau(\mathbf{w}^{(n)}|\mathbf{x}, \mathbf{z}, \mathbf{w}^{(n)})$. As proved below, this is equivalent to the optimization problem where the objective is to minimize the loss function defined as $\mathscr{L}_\tau(\mathbf{x}, \mathbf{z}) = -\ln p(Z|\mathbf{x}_{1:\tau})$. Then, the gradient of the loss function with respect to the input of the softmax layer is

$$\frac{\partial \mathscr{L}_\tau(\mathbf{x}, \mathbf{z})}{\partial a_k^t} = y_k^t - \frac{1}{p(Z|\mathbf{x}_{1:\tau})} \sum_{u \in B(\mathbf{z}, k)} \alpha(t, u)\beta_\tau(t, u), \tag{3.8}$$

where $p(Z|\mathbf{x}_{1:\tau})$ can be computed by

$$p(Z|\mathbf{x}_{1:\tau}) = \sum_{u=1}^{|\mathbf{z}'|} \alpha(\tau, u) \tag{3.9}$$

and the backward variable, $\beta_\tau(t,u)$, is initialized as

$$\beta_\tau(\tau, u) = 1, \ \forall u. \tag{3.10}$$

The new backward variable is propagated using the same recursion in (3.2), and the error gradients are computed with (3.5) as in the standard CTC algorithm.

Proof. In the maximization step, the objective is to obtain the derivative of $Q_\tau(\mathbf{w}|\mathbf{x},\mathbf{z},\mathbf{w}^{(n)})$ with respect to the input of the softmax layer, a_k^t, at time t. We first differentiate Q_τ with respect to y_k^t at $\mathbf{w} = \mathbf{w}^{(n)}$:

$$\left.\frac{\partial Q_\tau(\mathbf{w}|\mathbf{x},\mathbf{z},\mathbf{w}^{(n)})}{\partial y_k^t}\right|_{\mathbf{w}=\mathbf{w}^{(n)}} = \sum_{m=0}^{|\mathbf{z}|} p(\mathbf{z}_{1:m}|Z,\mathbf{x}_{1:\tau},\mathbf{w}^{(n)}) \frac{\partial \ln p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}{\partial y_k^t}. \tag{3.11}$$

With Bayes' rule, we obtain

$$p(\mathbf{z}_{1:m}|Z,\mathbf{x}_{1:\tau},\mathbf{w}^{(n)}) = \frac{p(\mathbf{z}_{1:m},Z|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}{p(Z|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})} = \frac{p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}{p(Z|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}, \tag{3.12}$$

and with simple calculus,

$$\frac{\partial \ln p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}{\partial y_k^t} = \frac{1}{p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})} \frac{\partial p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}{\partial y_k^t}. \tag{3.13}$$

Then, (3.11) becomes

$$\left.\frac{\partial Q_\tau(\mathbf{w}|\mathbf{x},\mathbf{z},\mathbf{w}^{(n)})}{\partial y_k^t}\right|_{\mathbf{w}=\mathbf{w}^{(n)}} = \frac{1}{p(Z|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}\sum_{m=0}^{|\mathbf{z}|}\frac{\partial p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}{\partial y_k^t} \tag{3.14}$$

$$= \frac{1}{p(Z|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}\frac{\partial p(Z|\mathbf{x}_{1:\tau},\mathbf{w}^{(n)})}{\partial y_k^t}. \tag{3.15}$$

If we define the loss function to be minimized as

$$\mathscr{L}_\tau(\mathbf{x},\mathbf{z}) = -\ln p(Z|\mathbf{x}_{1:\tau}), \tag{3.16}$$

then its derivative equals to (3.15) with the opposite sign:

$$\frac{\partial \mathscr{L}_\tau(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\left.\frac{\partial Q_\tau(\mathbf{w}|\mathbf{x},\mathbf{z},\mathbf{w}^{(n)})}{\partial y_k^t}\right|_{\mathbf{w}=\mathbf{w}^{(n)}}. \tag{3.17}$$

From now on, we drop $\mathbf{w}^{(n)}$ without loss of generality. Let

$$\beta_{\tau,m}(\tau,u) = \begin{cases} 1 & \text{if } u = 2m, 2m+1 \\ 0 & \text{otherwise} \end{cases}. \tag{3.18}$$

Following the standard CTC forward-backward equations in [49],

$$p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau}) = \sum_{u=1}^{|\mathbf{z}'|}\alpha(t,u)\beta_{\tau,m}(t,u) \tag{3.19}$$

$$\frac{\partial p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau})}{\partial y_k^t} = \frac{1}{y_k^t}\sum_{u\in B(\mathbf{z},k)}\alpha(t,u)\beta_{\tau,m}(t,u). \tag{3.20}$$

From (3.19) and (3.20), $p(Z|\mathbf{x}_{1:\tau})$ and its derivative become

$$p(Z|\mathbf{x}_{1:\tau}) = \sum_{m=0}^{|\mathbf{z}|} p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau}) = \sum_{u=1}^{|\mathbf{z}'|} \alpha(t,u)\beta_\tau(t,u) \tag{3.21}$$

$$\frac{\partial p(Z|\mathbf{x}_{1:\tau})}{\partial y_k^t} = \sum_{m=0}^{|\mathbf{z}|} \frac{\partial p(\mathbf{z}_{1:m}|\mathbf{x}_{1:\tau})}{\partial y_k^t} = \frac{1}{y_k^t} \sum_{u \in B(\mathbf{z},k)} \alpha(t,u)\beta_\tau(t,u), \tag{3.22}$$

where the new backward variable for $p(Z|\mathbf{x}_{1:\tau})$ is

$$\beta_\tau(t,u) = \sum_{m=0}^{|\mathbf{z}|} \beta_{\tau,m}(t,u), \tag{3.23}$$

which results in the simple initialization as

$$\beta_\tau(\tau,u) = 1, \ \forall u. \tag{3.24}$$

Then, the error gradients become

$$\frac{\partial \mathscr{L}_\tau(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\frac{1}{p(Z|\mathbf{x}_{1:\tau})} \frac{1}{y_k^t} \sum_{u \in B(\mathbf{z},k)} \alpha(t,u)\beta_\tau(t,u) \tag{3.25}$$

$$\frac{\partial \mathscr{L}_\tau(\mathbf{x},\mathbf{z})}{\partial a_k^t} = y_k^t - \frac{1}{p(Z|\mathbf{x}_{1:\tau})} \sum_{u \in B(\mathbf{z},k)} \alpha(t,u)\beta_\tau(t,u), \tag{3.26}$$

where

$$p(Z|\mathbf{x}_{1:\tau}) = \sum_{u=1}^{|\mathbf{z}'|} \alpha(\tau,u). \tag{3.27}$$

□

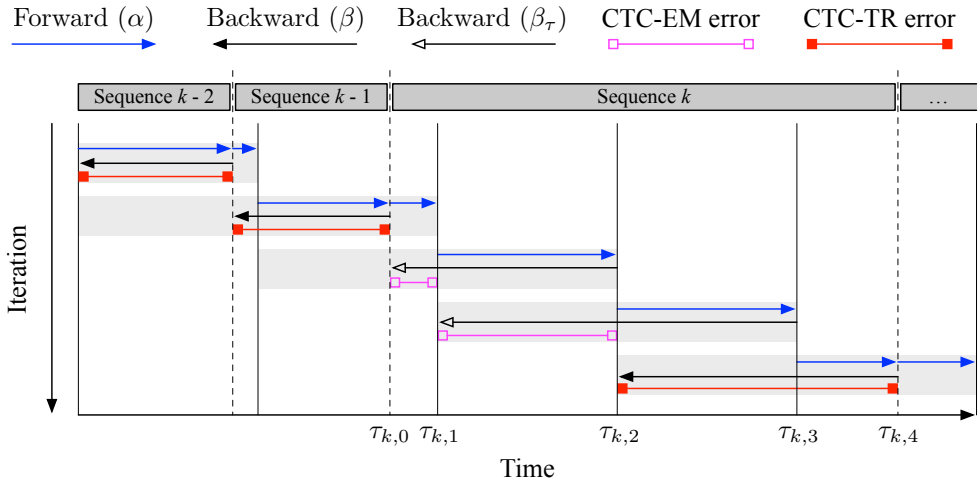


Figure 3.4: Online CTC($2h'$; h') training with a continuous stream of sequences. The shaded areas indicate the range of RNN unrolling, of which length is $2h'$, at each iteration. The modified segment boundaries for the sequence k, $\tau_{k,n}$, are shown.

3.4 Training Continuously Running RNNs

In this section, the proposed online CTC algorithm in Section 3.3 is extended for training infinitely long streams. The training stream can be naturally very long with the target sequence boundaries, or can be generated by concatenating training sequences in a certain order. When trained on this training stream without external reset of the RNN at the sequence boundaries, the resulting RNN can also continuously process infinitely long input streams without pre-segmentation or external reset. This property has been proved useful for the low-latency ISR, which will be described in Chapter 4, or spoken term detection systems [50] since we can remove the frontend voice activity detector [55] for detecting and pre-segmenting utterances.

The CTC(h; h') algorithm can directly be applied to the infinitely long training streams as shown in Figure 3.4. When the sequence boundaries are reached during the

forward activation, we perform CTC-TR, initialize the forward variable, and process the next sequence with some frame offset. Also, care should be taken on the transition of CTC labels at the boundary. Assume that the last label of the sequence k and the first label of the sequence $k+1$ are the same. Then, a CTC blank label should be inserted between two sequences since the same labels that occur consecutively in the decoding path are folded into one label. In practice, this folding can easily be prevented by forcing the blank label at the first frame of each sequence by modifying the initialization of the forward variable as follows:

$$\alpha_c(1,u) = \begin{cases} y_b^1 & \text{if } u = 1 \\ 0 & \text{otherwise} \end{cases}, \tag{3.28}$$

where the subscript c indicates the continuous CTC training.

3.5 Parallel Training

In a massively parallel shared memory model such as a GPU, efficient parallel training is achieved by making use of the memory hierarchy. For example, computing multiple frames together reduces the number of read operations of the network parameters from the slow off-chip memory by temporarily storing them on the on-chip cache memory and reuse them multiple times. For training RNNs on a GPU, this parallelism can be explored by employing multiple training sequences concurrently as explained in Chapter 2.

The continuous CTC(h; h') algorithm in Section 3.4 can be directly extended for parallel training with multiple streams. Since the forward step size and the unroll amount is fixed, the RNN forward, backward, gradient computation, and weight up-

date steps can be synchronized over multiple training streams. Thus, the GPU based parallelization approach in Chapter 2 can be employed for the RNN training. Although the computations in the CTC(h; h') algorithm are relatively fewer than those of the RNN, further speed-up can be achieved by parallelizing the CTC algorithm similarly.

3.6 Experiments

3.6.1 End-to-End Speech Recognition with RNNs

For the evaluation of the proposed approach, we present examples of end-to-end speech recognition with character-level RNN language models (LMs) and tree-based online decoding, where the details of the system will be described in Chapter 4. The acoustic RNN is a deep unidirectional long short-term memory (LSTM) network [56] with forget gates [27] and peephole connections [57], which is trained with the online CTC algorithm on the continuous stream of speech. Also, a character-level RNN language model [19] is employed for tree-based decoding. The system continuously recognizes infinitely-long input speech in realtime without pre-segmentation.

Specifically, the acoustic RNN has 3 unidirectional LSTM layers, where each layer has 768 LSTM cells. The output layer is a 31-dimensional softmax layer. Each unit of the softmax layer represents one of the posterior probabilities of 26 alphabet characters, two special characters (. and '), a whitespace character, the end of sentence (EOS) symbol, and the CTC blank label. The input of the network is a 123-dimensional vector that consists of a 40-dimensional log Mel-frequency filterbank feature vector plus energy, and their delta and delta-delta values. The feature vectors are extracted from the speech waveform in every 10 ms with 25 ms Hamming window

using HTK [58]. Before being fed into the RNN, feature vectors are element-wisely normalized to the zero mean and the unit standard deviation, where the statistics are extracted from the training set.

The character-level RNN LM consists of 2 unidirectional LSTM layers with 512 cells per layer. The input is a 30-dimensional one-hot encoded vector that represents a current label, and the output is the probabilities of the next labels. The input and output labels are same as the output labels of the acoustic RNN except the CTC blank label. The RNN LM considers the past and current inputs for computing the probabilities of the next labels.

3.6.1.1 Wall Street Journal (WSJ) Corpus

The experiments are performed on the Wall Street Journal (WSJ) [52] corpus. The RNN LM is trained with the text-based language model training data included in the WSJ corpus with the resulting bit-per-character (BPC) of 1.167. For the acoustic RNN training, the subset of the WSJ SI-284 set is used, where only the utterances with non-verbalized punctuations (NVPs) are included, resulting in about 71 hours of utterances. The histogram of the length of the sequences in the training set is shown in Figure 3.5. Note that the average length of the sequences is 772.5 frames. If we unroll the network over N frames, the sequences longer than N frames will not be fully covered by CTC-TR.

In Figure 3.6, the CTC-TR coverage is further analyzed with respect to the length of the sequence and the unroll amount. When the stream of sequences are trained with the continuos CTC algorithm, the CTC-TR coverage varies depending on the frame offsets of CTC(h; h'). The average coverage is calculated assuming that the offset is uniformly distributed. If the probability that a certain frame is included in the

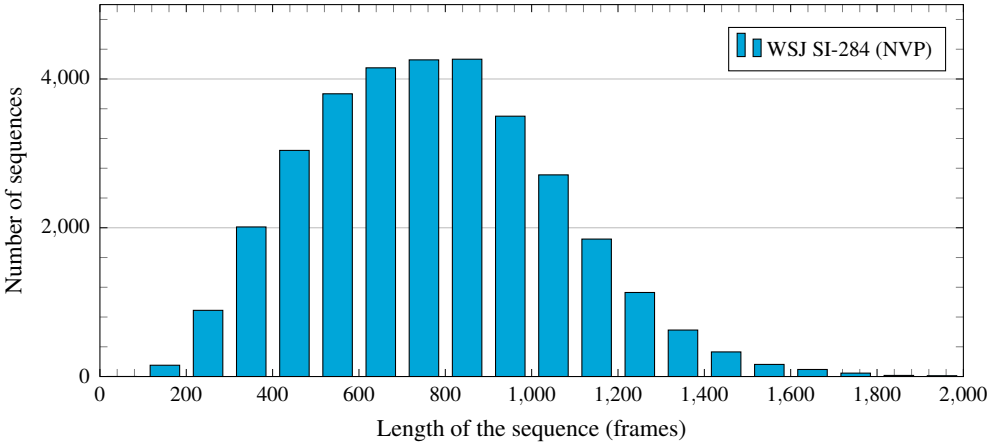


Figure 3.5: Histogram of the length of the sequences in the WSJ SI-284 training set, where only the utterances with non-verbalized punctuations (NVPs) are considered. The feature frames are extracted with the period of 10 ms.

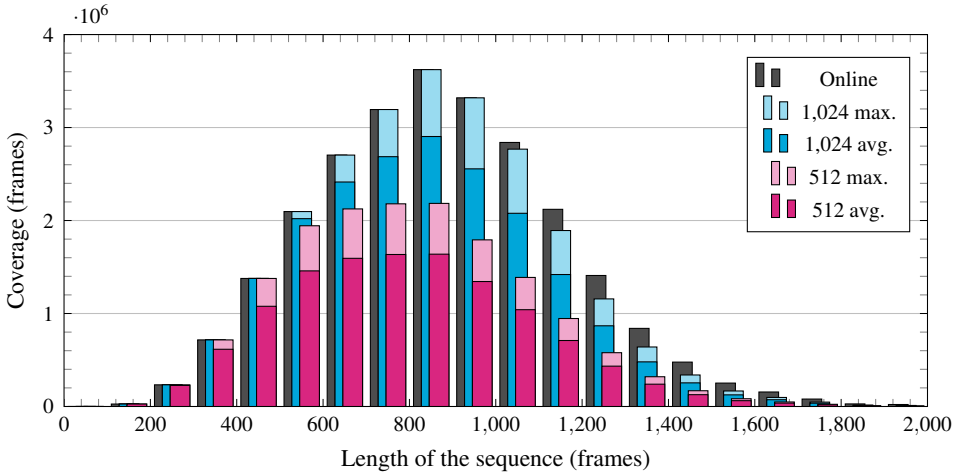


Figure 3.6: Coverage of the trainable frames with respect to the length of the sequences in the WSJ SI-284 (NVP) training set. The average and maximum coverages of CTC-TR on continuous training streams are visualized for the unroll amount of 512 and 1,024 when CTC($2h'$; h') is applied. Note that the proposed online CTC algorithm (CTC-TR + CTC-EM) covers the entire training frames.

coverage is greater than zero, then the frame is included in the maximum coverage. For the experiments, we only consider CTC($2h'$; h'), that is, the unroll amount is twice as much as the forward step size. Then, unrolling the network 1,024 times results in the CTC-TR coverage of 79.48 % on average and 95.69 % at maximum. On the other hand, when the unroll amount is 512, CTC-TR only covers 48.16 % on average and 63.27 % at maximum. Note that the full coverage is achieved when CTC-TR is combined with CTC-EM.

The WSJ Nov'93 20K development set and the WSJ Nov'92 20K evaluation set are used as the development (validation) set and the test (evaluation) set, respectively. For the final evaluation of the network after training, a single test stream is used that is generated by concatenating all of the 333 utterances in the test set.

3.6.1.2 Training Procedure

The RNN LM is trained with truncated BPTT(512; 256) on infinite training streams generated by concatenating sentences in the text training data. Note that the EOS symbols are inserted between sentences. The training is performed on a GPU with multiple streams. We applied ADADELTA [59] for annealing and early stopping for preventing overfitting. However, overfitting was not observed in our experiments.

The acoustic RNN are trained on a GPU as in Section 3.5 with the memory usage constraint. To maintain the memory usage the same while changing the unroll amount, we fixed the total amount of unrolling over multiple training streams to 16,384. For example, the number of parallel streams become 8 with the unroll amount of 2,048 and 32 with 512 times of unrolling. The total amount of GPU memory usage is about 9.5 GiB in our implementation.

The performance evaluation of the network is performed at every 10,485,760

training frames (i.e. N continuous training streams with the length of $10,485,760/N$ each) in terms of word error rate (WER) on the 128 parallel development streams of which the length is 16,384 each. For this intermediate evaluation, best path decoding [18] is employed without the RNN LM for fast computation.

For the online update of the RNN parameters, the stochastic gradient descent (SGD) method is employed and accelerated by the Nesterov momentum of 0.9 [60, 61]. Also, the network is annealed by combining the early stopping technique as follows. If the network performance based on the intermediate evaluation is not improved for 11 consecutive times (10 times of retry), the learning rate is reduced by the factor of 10 and the training is resumed from the second best network. The training starts from the learning rate of 10^{-5} and finishes when the learning rate becomes less than 10^{-7}.

The pre-trained network is used for CTC-TR and CTC-EM combined training because the expectation step of CTC-EM requires the RNN to align the target labels in a certain level. The pre-trained networks are obtained by early stopping the CTC-TR training of the networks when the performance is not improved during 6 consecutive intermediate evaluations using the learning rate of 10^{-5}. For the CTC-TR and CTC-EM combined training with the unroll amount of 512, 1,024, and 2,048, the training starts from the pre-trained network that is trained with the same amount of unrolling. Otherwise, for the combined training with the unrolling less than 512 times, we use the pre-trained network with the unroll amount of 512.

3.6.1.3 Evaluation

Figure 3.7 shows the convergence curves in terms of WER on the development set without the RNN LM using various unroll amounts and training algorithms, where

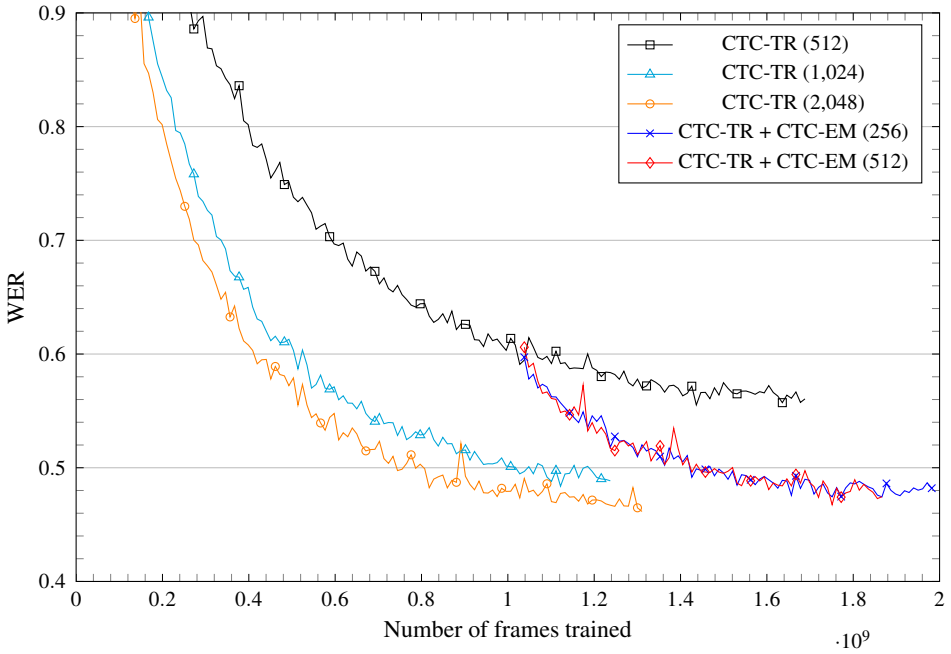


Figure 3.7: Convergence curves in terms of WER on the development set with the various unroll amounts of 256, 512, 1,024, and 2,048, and the fixed learning rate of 10^{-5}.

the unroll amount is twice the forward step size. The convergence speed of the CTC-TR only training decreases when the unroll amount becomes smaller. This is because the percentage of the effective training frames becomes smaller due to the reduced CTC-TR coverage. Also, it can be observed that the performance of the CTC-TR only trained network with 512 times of unrolling converges to the worse WER than those of the other networks due to the reduced size of the effective training set. On the other hand, the convergence curves of the CTC-TR and CTC-EM combined training with the unroll amounts of 256 and 512 are similar to that of the CTC-TR only training with 2,048 times of unrolling. Considering that the average sequence length of the training set is 772.5 frames, the results are quite encouraging.

Table 3.1: Comparison of the CTC-TR coverages, the CER and WERs on the test set, and the training speeds on the GPU with varying amounts of unrolling.

# Streams	**CTC-TR coverage (%)**		**CER / WER (%)**			**Training speed (frames/s)**	
× # Unroll	Average	Maximum	CTC-TR	+ CTC-EM	+ RNN LM	CTC-TR	+ CTC-EM
8 × 2,048	97.84	99.995	-	**10.6 / 38.4**	4.00 / 9.30	3.81 k	3.80 k
16 × 1,024	79.48	95.69	11.2 / 39.1	10.9 / 38.6	4.13 / 9.52	6.79 k	6.60 k
32 × 512	48.16	63.27	13.9 / 47.2	10.9 / 38.8	**3.89 / 8.88**	12.58 k	11.70 k
64 × 256	24.82	33.06	-	11.2 / 39.7	4.08 / 9.53	18.03 k	15.99 k
128 × 128	12.43	16.57	-	11.3 / 40.0	3.89 / 9.20	23.64 k	20.54 k
256 × 64	6.21	8.29	-	11.4 / 40.1	4.41 / 9.85	26.98 k	22.24 k

The evidence of the similar convergence curves with the different unroll amounts implies that the training can be accelerated under the memory usage constraint by employing more parallel training streams with less unrolling. To examine how much speed-up can be achieved on a GPU, further experiments are performed as in Table 3.1. The training speed is measured on the system equipped with NVIDIA GeForce Titan X GPU and Intel Xeon E5-2620 CPU. For the final character error rate (CER) and WER report on the test set, the output of the RNN is decoded by the tree-based online CTC beam search with and without language models. Note that the experiment with the unroll amount of 2048 is the baseline, where CTC-TR covers most of the training frames and there is little difference from the standard CTC training. As shown in the table, we can achieve a great amount of speedup without sacrificing much WERs. Also, it is possible to train a network with only 64 times of unrolling, which corresponds to 640 ms window, at the cost of 4.5% relative WER when decoded without the RNN LM.

The RNN LM is integrated with a beam width of 512, a beam depth of 50, an LM weight of 2.0, and an insertion bonus of 1.5. When the RNN LM is applied, the baseline network shows 9.30% WER. On the other hand, 8.88% WER is obtained

with the acoustic RNN trained with only 512 times of unrolling. However, we consider this improvement is due to the noise in the experimental results. It is observed that the early stopping of training based on the intermediate WER evaluation without the RNN LM does not guarantee the best performance when the decoding is performed with the RNN LM. Nevertheless, it seems there is a slight performance loss when the network is trained with only 64 times of unrolling. Note that 8.9% WER is achieved in Chapter 4 with the same network structure. Also, 8.7% and 7.34% WERs were reported in [10] and [45], respectively, with bidirectional RNNs for sentence-wise recognition. Our results in Table 3.1 is reported without any regularization techniques, such as weight noise in [10] or dropout [62]. For fair comparison, we also trained a unidirectional LSTM network with 4 layers, where each layer contains 512 cells, with online CTC(1024; 512) and dropout for RNNs [63]. This model achieves 32.5% WER without LMs, which is comparable to 30.1% WER obtained with a deep bidirectional LSTM network [10].

3.6.2 Phoneme Recognition on TIMIT

3.6.2.1 TIMIT Corpus

The TIMIT corpus [53] contains American English recordings of 630 speakers from 8 major dialect regions in the United States. The training set contains about 3.1 hours of 3,696 utterances from 462 speakers after removing the SA recordings, in which only two sentences are spoken by multiple speakers. Figure 3.8 shows the histogram of the length of the training sequences, where the feature frames are extracted with the 10 ms period. The average length of the training sequences is 304 frames. We use the *core test* set with 192 utterances as the test set. The development set contains the

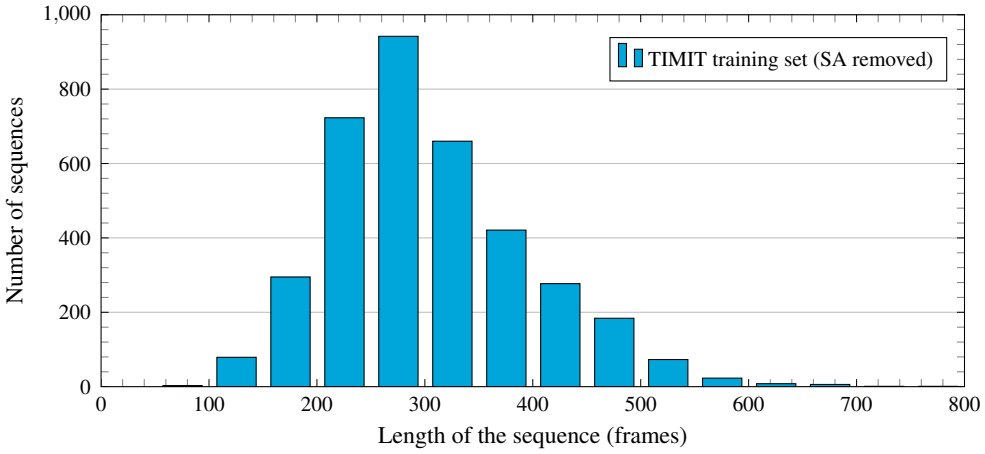


Figure 3.8: Histogram of the length of the sequences in the TIMIT training set (SA removed), where the feature frames are extracted with the period of 10 ms.

remaining 1,152 utterances that are obtained by excluding the *core test* set from the *complete test* set. The corpus also includes the full phonetic transcriptions.

3.6.2.2 Network Structure

The network structure is a deep unidirectional LSTM RNN with 3 LSTM hidden layers, where each LSTM layer has 512 cells. The input is the same log Mel-frequency filterbank feature as in the WSJ experiments. The training procedure is also similar. The original TIMIT transcriptions are based on 61 phonetic labels. Accordingly, the RNN output is a 62-dimensional vector that consists of the probabilities of the original 61 phonemes and the extra CTC label. However, after decoding, they are mapped to 39 phonemes for evaluation as in [64].

3.6.2.3 Training Procedure

For the experiments, the continuous CTC($2h'$; h') algorithm is employed so that the resulting RNN can run continuously on a infinitely long stream of the input speech. The networks are pre-trained with ADADELTA [59], where the local learning rates are adaptively adjusted using the statistics of the recent gradient values. Before the online CTC training with the unroll amount greater than of equal to 512, the pre-training is performed for the 8 M (8×2^{20}) training frames with the unroll amount of 2,048, the learning rate of 10^{-5}, the Nesterov momentum of 0.9, and the RMS decay rate of 0.99 for ADADELTA. On the other hand, we pre-trained the network with 12 M frames for the subsequent CTC training with less than 512 unroll steps. Unlike in the WSJ experiments, it is observed that applying the standard SGD method at the beginning often fails to initiate the training. We consider this is because the gradient computed by the SGD method is initially not noisy enough to help the parameters escape from the initial saddle point.

After the pre-training, the standard SGD is applied with the Nesterov momentum of 0.9. The training starts with the learning rate of 10^{-4}. The intermediate evaluations are performed at every 2 M (2×2^{20}) training frames on the development set with the best path decoding. If the phoneme error rate (PER) fails to improve during 6 consecutive evaluations, the learning rate decreases by the factor of 2 and the parameters are restored to those of the second best network. The training finishes when the learning rate becomes less than 10^{-6}.

The network is regularized with dropout [62] in both the pre-training and the main training stages following the approach in [63], that is, dropout is only applied on the non-recurrent connections. The dropout rate is fixed to 0.5 throughout the

Table 3.2: Comparison of CTC-TR coverages and PERs on the test set after CTC($2h'$; h') training with the varying amounts of unrolling.

# Streams	**CTC-TR coverage (%)**		**PER (%)**			
× # Unroll	Average	Maximum	Mean	± Stdev.	Min.	Max.
8 × 2,048	100.0	100.0	21.14	± 0.29	20.91	21.57
16 × 1,024	99.80	100.0	20.82	± 0.17	20.66	21.03
32 × 512	89.48	99.60	21.18	± 0.40	20.60	21.48
64 × 256	60.69	79.37	20.77	± 0.24	20.47	20.97
128 × 128	31.53	42.02	**20.73**	± 0.40	**20.39**	21.25
256 × 64	15.77	21.03	21.00	± 0.16	20.78	21.15

experiments.

3.6.2.4 Evaluation

The networks are evaluated on the very long test stream that is obtained by concatenating the entire test sequences. For the evaluation, the network output is decoded by the CTC beam search. The experiments are repeated 4 times and the mean and standard deviation estimates of PERs are reported based on the reduced 39-phoneme set.

The RNNs are unrolled 64, 128, 256, 512, 1,024, and 2,048 times. As shown in Table 3.2, the various unroll amounts make little difference to the final PERs on the test set. When the RNN is unrolled only 128 times, which is less than the average length of training sequences, the best PER of 20.73±0.40% is obtained. On the other hand, the training with the unroll amount of 2,048 results in slightly degraded performance since it becomes harder for RNNs to catch the dependencies between the input and output sequences due to the noisy input frames from the consecutive sequences.

The performance of the proposed online CTC algorithm is compared with the

Table 3.3: Comparison of the proposed online CTC algorithm and the other models in the literature in terms of PER on the test set.

Model	**Network (# param)**	**Bi-**	**Test sequence**	**PER (%)**
Proposed online CTC	LSTM (5.5 M)	**No**	**Almost infinite stream** [a]	20.73
Attention-based model [42]	Conv.[b]+GRU[c]	**Yes**	**Long sequences** [d] Utterance-wise	About 20 17.6
RNN transducer [39]	LSTM (4.3 M)	Yes	Utterance-wise	17.7
Sequence-wise CTC [39]	LSTM (3.8 M)	Yes No	Utterance-wise	18.4 19.6

[a] Generated by concatenating all of the 192 test utterances
[b] Convolutional features
[c] Gated recurrent unit [37]
[d] Generated by concatenating 11 utterances

other models in Table 3.3. The other models employ early stopping to prevent overfitting and add weight noise while training for regularization. The bidirectional attention-based model in [42] shows 17.6% PER with utterance-wise decoding. However, the PER increases to about 20% with the long test sequences that are generated by concatenating 11 utterances. On the other hand, our CTC(128; 64)-trained unidirectional RNNs show 20.73$\pm$0.40% PER with a very long test stream that is made by concatenating the entire 192 test utterances. Note that, unlike the CTC-trained unidirectional RNNs, the bidirectional models require unrolling in test time and have to listen the entire speech before generating outputs. Therefore, the proposed unidirectional RNN models are more suitable for realtime low-latency speech recognition systems without sacrificing much performance.

3.7 Concluding Remarks

Throughout the chapter, the online CTC(h; h') algorithm is proposed for sequence to sequence learning with unidirectional RNNs using partial windows. The algorithm consists of CTC-TR and CTC-EM. CTC-TR is the standard CTC algorithm with truncation and CTC-EM is the generalized EM based algorithm that covers the training frames that CTC-TR cannot be applied. The proposed algorithm allows the unroll amount to be less than the length of the training sequence and is suitable for small footprint online learning systems or massively parallel implementation on a shared memory model such as a GPU. Also, the online CTC algorithm is extended for training continuously running RNNs without external reset, and evaluated in the WSJ and TIMIT experiments. On the WSJ corpus, when the memory capacity is constrained, the proposed approach achieves significant speed-up on a GPU without sacrificing the performance of the resulting RNN much. We expect that further acceleration of training will be possible with lower performance loss when different unroll amounts are used in the pre-training, main training, and annealing stages.

Chapter 4

Character-Level Incremental Speech Recognition

4.1 Introduction

Incremental speech recognition (ISR) allows a speech-based interaction system to react quickly while the utterance is being spoken. Unlike offline sentence-wise automatic speech recognition (ASR), where the decoding result is available after a user finishes speaking, ISR returns N-best decoding results with small latency during speech. These N-best results, or hypotheses, gradually improve as the system receives more speech data. Since ISR is usually employed for immediate reaction to speech, word stability [13, 14] and incremental lattice generation [12] have been important topics.

In this chapter, we introduce an end-to-end character-level ISR system with two unidirectional recurrent neural networks (RNNs). An acoustic RNN roughly dictates the input speech and an RNN-based language model is employed to augment the

dictation result through decoding. Compared to a conventional word-level backend for speech recognition system, the character-level ASR is capable of dictating out of vocabulary (OOV) words based on the pronunciation. Also, our model is trained directly from speech and text corpus and does not require external word dictionary or senone modeling.

There have been efforts to deal with OOV words in conventional HMM based ASR systems. In [65], graphemes are employed as basic units instead of phonemes. Also, a sub-lexical language model is proposed in [66] for detecting previously unseen words.

RNN-based character-level end-to-end ASR systems were studied in [10, 40, 67, 45, 41]. However, they lack the capability of dictating OOV words since the decoding is performed with word-level LMs. Recently, a lexicon-free end-to-end ASR system is introduced in [44], where a character-level RNN LM is employed. We further improve this approach by employing prefix tree based online beam search with additional depth-pruning for ISR.

The character-level ISR system proposed in this chapter is composed of an acoustic RNN and an RNN LM. The acoustic RNN is end-to-end trained with connectionist temporal classification (CTC) [18] using Wall Street Journal (WSJ) speech corpus [52]. The output of the acoustic RNN is the probability of characters, which are decoded with character-level beam search to generate N-best hypotheses. To improve the performance, a character-level RNN LM is employed to augment the beam search performance. Also, we propose depth-pruning for efficient tree-based beam search. The RNN LM is separately trained with a large text corpus that is also included in WSJ corpus. Unlike for word-level language modeling, conventional statistical LMs such as n-gram back-off models cannot be used because much longer history window

is required for character-level prediction. Both acoustic RNN and RNN LM have deep unidirectional long short-term memory (LSTM) network structures [56, 68]. For continuous ISR on infinitely long input speech, they are trained with virtually infinite training data streams that are generated by randomly concatenating training sequences.

The proposed model is evaluated on a single test sequence that is generated by concatenating all test utterances in WSJ `eval92` (Nov'92 20k evaluation set) without any external reset of RNN states at the utterance boundaries. The ISR performance is examined by varying the beam width and depth. Generally, wider beam increases the accuracy. Under the same beam width, there is a trade-off between the accuracy and stability (or latency), where the balance between them can be adjusted by the beam depth.

The chapter is organized as follows. In Section 4.2, we describe RNN-based speech and language models. Section 4.3 contains the character-level beam search algorithm that is augmented by the RNN LM. Our ISR system is evaluated in Section 4.4. Concluding remarks follow in Section 4.5.

4.2 Models

4.2.1 Acoustic Model

The acoustic model is a deep RNN trained with CTC [18]. The network consists of three LSTM layers with 768 cells each, where the network has total 12.2 M trainable parameters. The model is similar to the one in the previous work about end-to-end speech recognition with RNNs [10] except a few major differences. In our case, the RNN is trained by online CTC, which is proposed in Chapter 3, with very long train-

ing sequences that are generated by randomly concatenating several utterances. There is no need to reset the RNN states at the utterance boundary. This is necessary for ISR systems that runs continuously with an infinite input audio stream. Also, our model has a unidirectional structure since bidirectional networks that are usually employed for end-to-end speech recognition are not suitable for low-latency speech recognition. This is because the backward layers in the bidirectional networks cannot be computed before the input utterance is finished.

The input of the network is a 40-dimensional log mel-frequency filterbank feature vector with energy and their delta and double-delta values, resulting in an 123-dimensional vector. The feature vectors are extracted every 10 ms with 25 ms Hamming window. The input vectors are element-wisely standardized based on the statistics obtained from the training set. The output is a 31-dimensional vector that consists of the probabilities of 26 upper case alphabets, 3 special characters, the end-of-sentence (EOS) symbol, and the CTC blank label.

The networks are trained with stochastic gradient descent (SGD) with 8 parallel input streams on a GPU using the system described in Chapter 2. The networks are unrolled 2,048 times and weight updates are performed every 1,024 forward steps. The network performances are evaluated at every 10 M training frames. The evaluation is performed on total 2 M frames from the development set. The learning rate starts from 1×10^{-5} and is reduced by the factor of 10 whenever the WER on the development set is not improved for 6 consecutive evaluations. The training ends when the learning rate drops below 1×10^{-7}.

We trained the networks on two training sets. The first one is the standard WSJ `SI-284` set and the second one, `SI-ALL`, is the set of all speaker independent training utterances in the WSJ corpus. Note that the utterances with verbalized punctuations

are removed from both training sets. Also, odd transcriptions are filtered out, which makes the final `SI-284` and `SI-ALL` sets contain roughly 71 and 167 hours of speech, respectively. WSJ `dev93` (Nov'93 20k development set) and `eval92` (Nov'92 20k evaluation set) sets are used as the development set and the evaluation set, respectively.

4.2.2 Language Model

An RNN language model (LM) [26] is employed for the proposed ISR system since conventional statistical LMs such as n-gram back-off models are not suitable for character-level prediction since they cannot make use of very long history windows. Specifically, the RNN LM has a deep LSTM network structure with two LSTM layers where each of them has 512 memory cells, resulting in total 3.2 M parameters.

The input of the RNN LM is a 30-dimensional vector, where the current label (character) is one-hot encoded. The output is also a 30-dimensional vector which represents the probabilities of next labels. Although the RNN LM is trained to predict the next characters with only the current character as the input, the past character histories are internally stored inside the RNN and used for the prediction. It is well known that RNN LM can remember contexts for very long time steps.

As for the acoustic RNN, the RNN LM is trained on a very long text stream that is generated by attaching randomly picked sentences and inserting EOS labels between sentences. The RNN LM is trained with AdaDelta [59] based SGD method for accelerated training and better annealing. The WSJ LM training text with non-verbalized punctuation, which contains about 215 M characters, is used for training the RNN LM. Randomly selected 1% of the corpus is reserved for evaluation, on which the final bits-per-character (BPC) of the RNN LM is 1.167 (character-level

```
THREE ISSUES ADVANCED MICRO OF AMERICA THE ONLY WAY TO DIVERSIFY INTO
TREATING MODERN ARMIES

LOOKING AHEAD TO MR. LEYSEN WITH AN INTOLERABLE POP CUT WHEN AN ALL
POWERFUL STUDENT SEEKS ITS CORE DRIVING UPJOHN STOVES

AMERICAN EXPRESS HASN'T YET SWORED PARTICULARLY WITH THE RESTRUCTURING IS
A COMMITMENT TO BUY POTENTIAL BUYERS IN THE OPEN MARKET

THE WHITE HOUSE GESTURES CLEAR EMPHASIS WITH NO REASON FOR INSTEAD ABROAD
HE CHANGED TO WHAT WAS DROPPED BY HIM
```

Figure 4.1: Example of character-level random text generation with the RNN LM.

perplexity of 2.245).

Random sentences can be generated following the method described in [19]. Briefly, the next label is randomly picked following the probabilities of the current output of the RNN LM and fed back to the RNN in the next step. By iterating these steps, texts can be sequentially generated as shown in Figure 4.1. From the example, it is clear that the RNN LM learned the linguistic structures as well as spellings of words that frequently appear.

4.3 Character-Level Beam Search

4.3.1 Prefix-Tree-Based CTC Beam Search

Let L be the set of labels without the CTC blank label. The label sequence $\mathbf{z}$ is a sequence of labels in L. The length of the label sequence $\mathbf{z}$ is less than or equal to the number of input frames. The objective of the beam search decoding is to find the label sequence that has the maximum posterior probability given the input features

from time 1 to t generated by the acoustic RNNs, that is,

$$\mathbf{z}_{\max} = \arg\max_{\mathbf{z}} P(\mathbf{z}|x_{1:t}), \tag{4.1}$$

where $x_{1:t}$ is the input features from time 1 to t.

However, the CTC-trained RNN output has one more blank label. Let L' be the set of labels (or CTC states) with the additional CTC blank label, and the path $\pi_t^{(i)}$ be a sequence of labels in L' from time 1 to t. The length of the path $\pi_t^{(i)}$ is the same as t. By the definition of CTC, every π can be reduced into the corresponding $\mathbf{z}$. For example, π with "aab-c–a" corresponds to $\mathbf{z}$ with "abca", where "-" is the blank label.

There can be many paths, $\pi_t^{(i)}$, that can be reduced into the same $\mathbf{z}$. Let $\mathscr{F}(\cdot)$ be a function that maps a path to the corresponding label sequence, that is, $\mathscr{F}(\pi_t^{(i)}) = \mathbf{z}$, then the posterior probability in (4.1) becomes,

$$P(\mathbf{z}|x_{1:t}) = \sum_{\{\forall i|\mathscr{F}(\pi_t^{(i)})=\mathbf{z}\}} P(\pi_t^{(i)}|x_{1:t}). \tag{4.2}$$

Therefore, if the two different paths $\pi_t^{(j)}$ and $\pi_t^{(k)}$ in the decoding network are mapped to the same $\mathbf{z}$, then they can be merged by summing their probabilities.

For the beam search, we first represent the lattice with a tree-based structure so that each node has one of labels in L as depicted in Figure 4.2. Then, backtracking from any node generates a unique label sequence $\mathbf{z}$. To deal with CTC state transitions, we need a state-based network that is represented with CTC states, L'. As shown in Figure 4.3, this can be easily done by expanding each tree node, of which label is in L, into two CTC states, one with the corresponding label in L' followed by the blank CTC label. Since the label-level (L) search network is based on a tree struc-

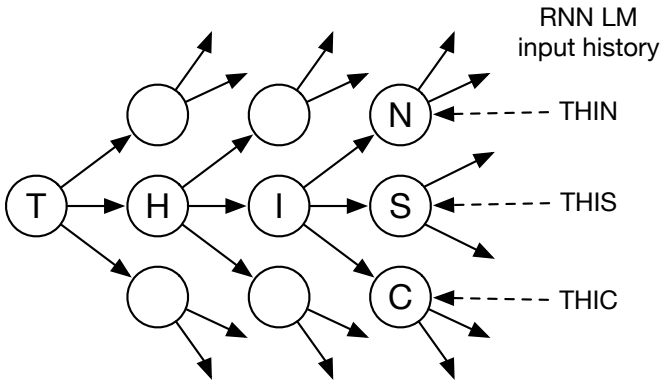


Figure 4.2: Beam search tree consisting of label nodes. The CTC blank label is not included.

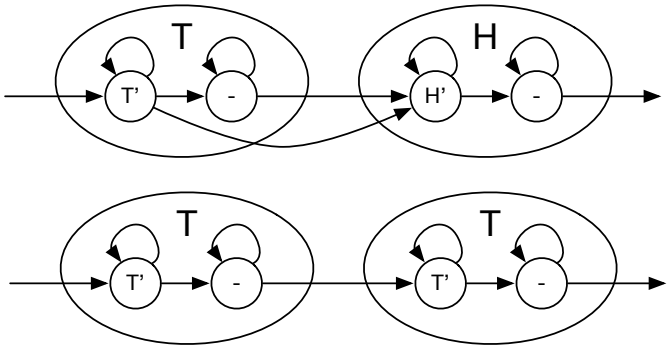


Figure 4.3: CTC state transition between two label nodes. If the two nodes have the same label, then a transition between the same CTC state is not allowed.

ture, two different state-level (L') paths with different label sequences never meet each other. This simplifies the problem since there is no interaction between two different sequence labelings (hypotheses) and (4.2) is the only equation that we should concern.

As proposed in [67, 44], external language models can be integrated by modifying the posterior probability term in (4.1) into:

$$\log(P(\mathbf{z}|x_{1:t})) = \log(P_{\mathrm{CTC}}(\mathbf{z}|x_{1:t})) + \alpha\log(P_{\mathrm{LM}}(\mathbf{z})) + \beta|\mathbf{z}|, \tag{4.3}$$

where α is the LM weight and β is the insertion bonus. This modification can be applied by adding the additional terms with α and β to the log probability of the destination state when a state transition between two different label nodes occurs.

The probability of the next label is computed using the RNN LM when a new active label node is added to the beam search tree. For this, the RNN LM context (hidden activations) is copied from the parent node to the child node and the RNN LM processes the new label of the child node with the copied context. Therefore, each active node has its own RNN LM context.

4.3.2 Pruning

Pruning of the search tree is performed by the standard beam search approach. That is, at each frame, only the active nodes with the top N hypotheses and their ancestor nodes remain alive after the pruning with the beam width of N. However, this standard pruning, or *width-pruning*, cannot prevent the tree from growing indefinitely especially when the input speech is very long. This gradually degrades the efficiency

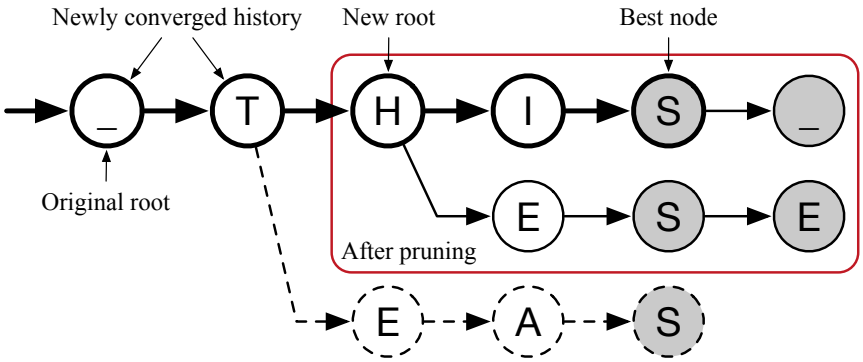


Figure 4.4: Example of depth-pruning with the beam depth of 2. The pruning is performed by selecting a new root node so that the new depth of the best hypothesis node becomes the beam depth. The shaded nodes indicate the original active nodes. Also, the path of the best hypothesis is drawn with thick strokes.

of beam search on recent nodes since more and more hypotheses would be wasted to maintain the old part of the lattice that is already out of the context range of the RNN LMs.

To remedy this issue, we propose an additional pruning method called *depth-pruning*. The procedure is as follows. First, find the M-th ancestor of the node with the best hypothesis, where M is the beam depth. Then, the ancestor node becomes a new root node. The pruning is performed by removing the nodes that are not descendants of the new root node. In this way, a beam can be better utilized for recent hypotheses rather than older ones. Figure 4.4 shows an example of depth-pruning with the beam depth of 2. Note that the depth of some nodes can be larger than the beam depth. In the following experiments, depth-pruning is performed every 20 frames.

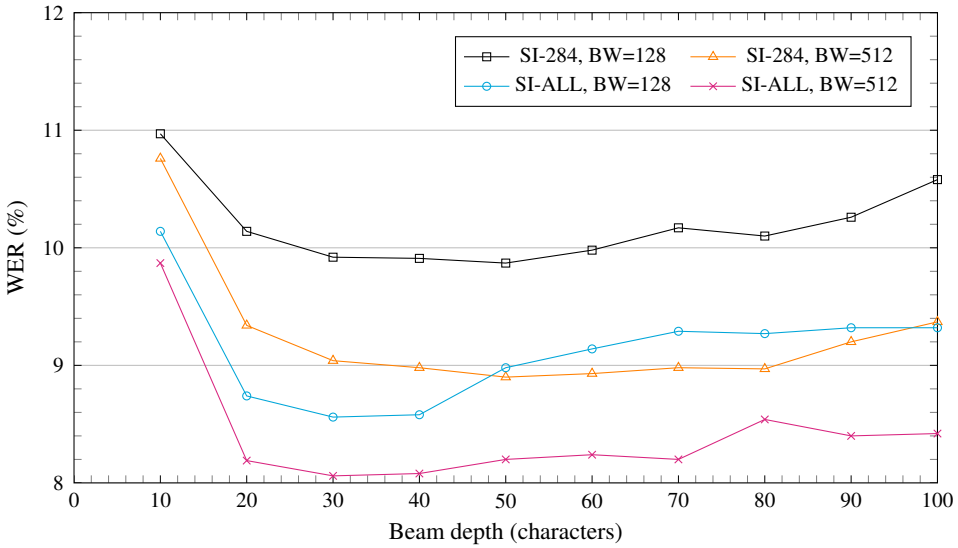


Figure 4.5: WER of the proposed online decoding on the evaluation set with respect to the beam depth. Experiments are conducted with two acoustic RNNs trained on `SI-284` and `SI-ALL` and beam search is performed with the beam width (BW) of 128 and 512.

4.4 Experiments

The proposed ISR system is evaluated on a single 42-minute speech stream that is formed by concatenating all 333 utterances in the evaluation set, `eval92` (WSJ Nov'92 20k evaluation set). We use $\alpha = 2.0$ and $\beta = 1.5$ for the system trained with `SI-284`, and $\alpha = 1.5$ and $\beta = 2.0$ for the other one trained with `SI-ALL`.

The effects of beam depth and width to the final WER are examined in Figure 4.5. The gap between the beam width of 128 and 512 is roughly 0.5% to 1% WER. However, there was little difference when the beam width increases from 512 to 2048 in our preliminary experiments. The best performing beam depths are 50 and 30 for the `SI-284` and `SI-ALL` systems, respectively. This means the `SI-ALL` system can

Table 4.1: CER / WER in percent on the evaluation set with online depth-pruning and offline sentence-wise decoding. The error rates are reported with two acoustic RNNs trained on SI-284 (71 hrs) and SI-ALL (167 hrs).

Method	Beam width	SI-284	SI-ALL
Online (no LM)	512	10.96 / 38.37	9.66 / 35.44
Online	128	4.25 / 9.87	3.56 / 8.56
Online	512	3.80 / **8.90**	3.39 / **8.06**
Sentence-wise	128	4.46 / 10.30	3.63 / 8.84
Sentence-wise	512	4.04 / 9.45	3.38 / 8.28

recognize speech more immediately than the SI-284 system. We consider this is because the acoustic model of the SI-ALL system can embed stronger language model due to increased training data, and can make decision more precisely without relying on the external language model much. The character error rate (CER) and WER are reported in Table 4.1 with the optimal beam depths. For comparison, we also report sentence-wise offline decoding results without depth-pruning.

The proposed ISR system is compared with other end-to-end word-level speech recognition systems in Table 4.2. The other systems perform sentence-wise offline decoding with bidirectional RNNs. The best result was achieved by Miao *et al*. [45] with a CTC-trained deep bidirectional LSTM network and a retrained trigram LM with extended vocabulary. The systems with the original trigram model provided with the WSJ corpus perform worse than our ISR system with character-level RNN LM. On the other hand, our system is beaten by the other ones with extended trigram models. However, more precise comparison of the decoding stages should be done by employing the same CTC model.

Figure 4.6 shows the incremental speech recognition result with the proposed ISR system. The best hypothesis is reported every 50 frames (500 ms). It is shown that the

Table 4.2: Comparison of WERs with other end-to-end speech recognizers in the literature. For reference, WERs of phoneme based GMM/DNN-HMM systems are also reported. All systems are trained with `SI-284` and evaluated on `eval92`.

System	Model	WER
Proposed ISR	Uni. CTC + Char. RNN LM	8.90%
Graves and Jaitly [10]	CTC + Trigram (extended)	8.7%
Miao *et al*. [45]	CTC + Trigram (extended)	7.34%
Miao *et al*. [45]	CTC + Trigram	9.07%
Hannun *et al*. [67]	CTC + Bigram	14.1%
Bahdanau *et al*. [41]	Encoder-decoder + Trigram	11.3%
Woodland *et al*. [69]	GMM-HMM + Trigram	9.46%
Miao *et al*. [45]	DNN-HMM + Trigram	7.14%

100: HE'S_THE_
150: HE'S_THE_ONLY_GU
200: HE'S_THE_ONLY_GUY_WHO_COULD_S
250: HE'S_THE_ONLY_GUY_WHO_COULD_SHOW_UP_IN_THE_
300: HE'S_THE_ONLY_GUY_WHO_COULD_SHOW_UP_IN_THE_PLAZA_I
350: ...PLAZA_IN_ROCK_R
400: ...PLAZA_IN_DRAW_RATE_OF_SEVE
450: ...PLAZA_IN_DRAW_RATE_OF_SEVENTY_FIVE_THO
500: ...PLAZA_AND_DRAW_CROWD_OF_SEVENTY_FIVE_THOUSAND_PEO
550: ...PLAZA_AND_DRAW_CROWD_OF_SEVENTY_FIVE_THOUSAND_PEOPLE_S
600: ...PLAZA_AND_DRAW_CROWD_OF_SEVENTY_FIVE_THOUSAND_PEOPLE_SAYS_ONE_LA
650: ...PLAZA_AND_DRAW_CROWD_OF_SEVENTY_FIVE_THOUSAND_PEOPLE_SAYS_ONE_LATIN_DIPLOM
700: ...PLAZA_AND_DRAW_CROWD_OF_SEVENTY_FIVE_THOUSAND_PEOPLE_SAYS_ONE_LATIN_DIPLOMAT
Ground truth: HE'S_THE_ONLY_GUY_WHO_COULD_SHOW_UP_IN_THE_PLAZA_AND_DRAW_
A_CROWD_OF_SEVENTY_FIVE_THOUSAND_PEOPLE_SAYS_ONE_LATIN_DIPLOMAT

Figure 4.6: Example of ISR partial results. The best hypothesis is shown at every 50 frames (500 ms). The word "ROCK" is corrected to "DRAW" after hearing "RATE" and "IN DRAW RATE" to "AND DRAW CROWD" while hearing "PEOPLE".

past best result can be corrected by making use of the additional speech input. For example, the word "ROCK" is changed to "DRAW" in the frame 450 by listening the word "RATE". Moreover, the correction of "IN DRAW RATE" to "AND DRAW CROWD" during hearing the word "PEOPLE" in the frame 500 is a good evidence that long term context can also be considered.

4.5 Concluding Remarks

A character-level incremental speech recognizer is proposed and analyzed throughout the chapter. The proposed system combines a CTC-trained RNN with a character-level RNN LM through tree-based beam search decoding. For online decoding with very long input speech, depth-pruning is proposed to prevent indefinite growth of the search tree. When the proposed model is trained with WSJ `SI-284`, 8.90% WER can be achieved on the very long speech that is formed by concatenating all utterances in the WSJ `eval92` evaluation set. The incremental recognition result shows the evidence that character-level RNN LM can learn dependencies between two words even when they are five words apart, which are hard to be caught using conventional n-gram back-off language models.

Note that the proposed system only requires speech and text corpus for training. External lexicon or senone modeling is not needed for training, which is a huge advantage. Moreover, it is expected that OOV words or infrequent words such as names of places or people can be dictated as they are pronounced.

Chapter 5

Character-Level Language Modeling with Hierarchical RNNs

5.1 Introduction

Language models (LMs) show the probability distribution over sequences of words or characters, and they are very important for many speech and document processing applications including speech recognition, text generation, and machine translation [1, 19, 70]. LMs can be classified into character-, word-, and context-levels according to the unit of the input and output. In the character-level LM (CLM) [19], the probability distribution of the next characters are generated based on the past character sequences. Since the number of alphabets is small in English, for example, the input and output of the CLM is quite simple. However, the word-level LM (WLM) is usually needed because the character-level modeling is disadvantaged in utilizing the long period of past sequences. However, the problem of the word-level model is the complexity of the input and output because the vocabulary size to be supported can

be bigger than 1 million.

LMs have long been developed by analyzing a large amount of texts and storing the probability distribution of word sequences into the memory. The statistical language model demands a large memory space, often exceeding 1 GB, not only because the vocabulary size is large but also their combinations needs to be considered. In recent years, the language modeling based on recurrent neural networks (RNNs) have been actively investigated [71, 72]. However, the RNN based WLMs still demand billions of parameters because of the large vocabulary size.

In this chapter, we propose hierarchical RNN based LMs that combine the advantageous characteristics of both character- and word-level LMs. The proposed network consists of a low-level and a high-level RNNs. The low-level RNN employs the character-level input and output, and provides the short-term embedding to the high-level RNN that operates as the word-level RNN. The high-level RNN do not need complex input and output because it receives the character-embedding information from the low-level network, and sends the word-prediction information back to the low-level in a compressed form. Thus, when considering the input and output, the proposed network is a CLM, although it contains a word-level model inside. The low-level module operates with the character input clock, while the high-level one runs with the space (`<w>`) and sentence boundary tokens (`<s>`) that separates words. We expect this hierarchical LM can be extended for processing a longer period of information, such as sentences, topics, or other contexts.

This chapter is organized as follows. Section 5.2 describes the background on character-level language modeling using RNNs and related work. RNN modeling including the external clock and reset signals is shown in Section 5.3, and the proposed language model using a hierarchical RNN is presented in Section 5.4. Section 5.5

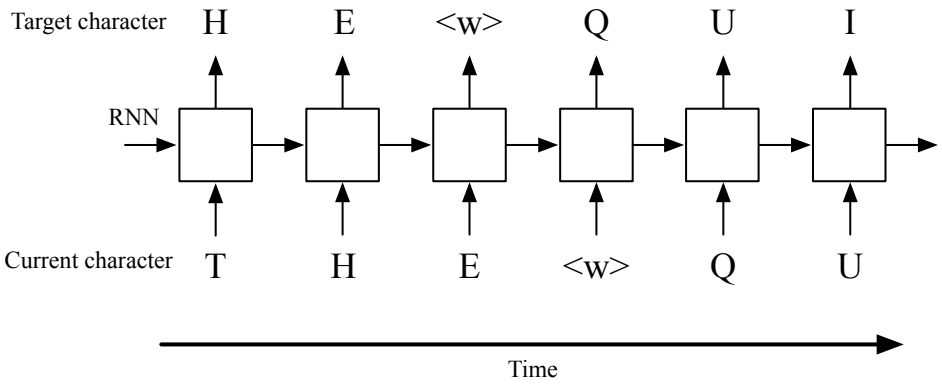


Figure 5.1: Training an RNN-based CLM.

gives the experimental results, and concluding remarks are given in Section 5.6.

5.2 Related Work

5.2.1 Character-Level Language Modeling with RNNs

CLMs need to consider longer sequence of history tokens to predict the next token than the WLMs, due to the smaller unit of tokens. Therefore, traditional N-gram models cannot be employed for CLMs. Thanks to the recent advances in RNNs, RNN-based CLMs has begun to show satisfactory performances [19, 73]. Especially, deep long short-term memory (LSTM) [56] based CLMs show excellent performance and successfully applied to end-to-end speech recognition system proposed in Chapter 4.

For training RNN CLMs, training data should be first converted to the sequence of one-hot encoded character vectors, $\mathbf{x}_t$, where the characters include word boundary symbols, `<w>` or space, and optionally sentence boundary symbols, `<s>`. Then, as

shown in Figure 5.1, the RNN is trained to predict the next character $\mathbf{x}_{t+1}$ by minimizing the cross-entropy loss of the softmax output [54] that represents the probability distributions of the next character.

5.2.2 Character-Aware Word-Level Language Modeling

There has been many attempts to make WLMs understand character-level inputs. One of the most successful approaches is to encode the arbitrary character sequence to fixed dimensional vector, which is called word embedding, and feed this vector to the word-level RNN LMs. In [74], convolutional neural networks (CNNs) are used to generate word embeddings, and achieve the state of the art results on English Penn Treebank corpus [75]. The similar CNN-based embedding approach is used by [72] with very large LSTM networks on the One Billion Word Benchmark [76], also achieving the state of the art perplexity. In [77, 78], bidirectional LSTMs are employed instead of CNNs for word embedding. However, in all of these approaches, LMs still generate the output probabilities at the word-level. Although the character-level modeling approach of the output word probability is introduced using CNN softmax in [72], the base LSTM network still runs with a word-level clock.

Our approach is different from the above ones in many ways. First, our base model is the character-level RNN LMs, instead of WLMs, and we extend this model to enhance the model to consider long-term contexts. Therefore, the output probabilities are generated with a character-level clocks. This property is extremely useful for character-level beam search for end-to-end speech recognition introduced in Chapter 4. Also, the input and output of our model are the same as those of the traditional character-level RNNs, thus the same training algorithm and recipe can be used without any modifications. Furthermore, the proposed models have significantly

less number of parameters compared to WLM-based ones, since the input and output complexity of our model does not directly depend on the vocabulary size of the training set. Note that a similar hierarchical concept has been used for character-level machine translation [79]. However, we propose more general hierarchical unidirectional RNN architecture that can be applied for various applications.

5.3 RNNs with External Clock and Reset Signals

In this section, we generalize the existing RNN structures and extend them with external clocks and reset signals. The extended models become the basic building blocks of the hierarchical RNNs.

Most types of RNNs or recurrent layers can be generalized as

$$\mathbf{s}_t = f(\mathbf{x}_t, \mathbf{s}_{t-1}) \ , \ \ \mathbf{y}_t = g(\mathbf{s}_t) \tag{5.1}$$

where $\mathbf{x}_t$ is the input, $\mathbf{s}_t$ is the state, $\mathbf{y}_t$ is the output at time step t, $f(\cdot)$ is the recurrence function, and $g(\cdot)$ is the output function. For example, a hidden layer of Elman networks [9] can be written as

$$\mathbf{y}_t = \mathbf{s}_t = \mathbf{h}_t = \sigma(W_{hx}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{5.2}$$

where $\mathbf{h}_t$ is the activation of the hidden layer, $\sigma(\cdot)$ is the activation function, W_{hx} and W_{hh} are the weight matrices and $\mathbf{b}_h$ is the bias vector.

LSTMs [56] with forget gates [27] and peephole connections [57] can also be converted to the generalized form. The forward equations of the LSTM layer are as

follows:

$$\mathbf{i}_t = \sigma(W_{ix}\mathbf{x}_t + W_{ih}\mathbf{h}_{t-1} + W_{im}\mathbf{m}_{t-1} + \mathbf{b}_i) \tag{5.3}$$

$$\mathbf{f}_t = \sigma(W_{fx}\mathbf{x}_t + W_{fh}\mathbf{h}_{t-1} + W_{fm}\mathbf{m}_{t-1} + \mathbf{b}_f) \tag{5.4}$$

$$\mathbf{m}_t = \mathbf{f}_t \circ \mathbf{m}_{t-1} + \mathbf{i}_t \circ \tanh(W_{mx}\mathbf{x}_t + W_{mh}\mathbf{h}_{t-1} + \mathbf{b}_m) \tag{5.5}$$

$$\mathbf{o}_t = \sigma(W_{ox}\mathbf{x}_t + W_{oh}\mathbf{h}_{t-1} + W_{om}\mathbf{m}_t + \mathbf{b}_o) \tag{5.6}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{m}_t) \tag{5.7}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ are the input, forget, and output gate values, respectively, $\mathbf{m}_t$ is the memory cell state, $\mathbf{h}_t$ is the output activation of the layer, $\sigma(\cdot)$ is the logistic sigmoid function, and $\circ$ is the element-wise multiplication operator. These equations can be generalized by setting $\mathbf{s}_t = [\mathbf{m}_t, \mathbf{h}_t]$ and $\mathbf{y}_t = \mathbf{h}_t$.

Any generalized RNNs can be converted to the ones that incorporate an external clock signal, c_t, as

$$\mathbf{s}_t = (1 - c_t)\mathbf{s}_{t-1} + c_t f(\mathbf{x}_t, \mathbf{s}_{t-1}) \ , \ \ \mathbf{y}_t = g(\mathbf{s}_t) \tag{5.8}$$

where c_t is 0 or 1. The RNN updates its state and output only when $c_t = 1$. Otherwise, when $c_t = 0$, the state and output values remain the same as those of the previous step.

The reset of RNNs is performed by setting $\mathbf{s}_{t-1}$ to 0. Specifically, (5.8) becomes

$$\mathbf{s}_t = (1 - c_t)(1 - r_t)\mathbf{s}_{t-1} + c_t f(\mathbf{x}_t, (1 - r_t)\mathbf{s}_{t-1}) \tag{5.9}$$

where the reset signal $r_t = 0$ or 1. When $r_t = 1$, the RNN forgets the previous contexts.

If the original RNN equations are differentiable, the extended equations with

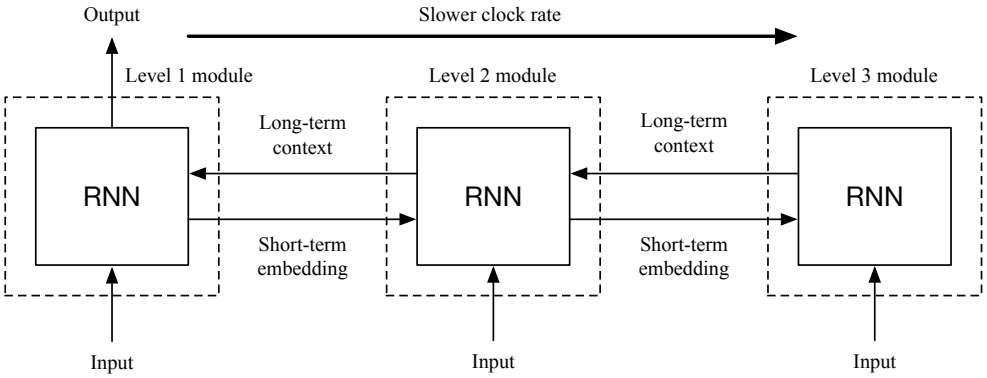


Figure 5.2: Hierarchical RNN (HRNN).

clock and reset signals are also differentiable. Therefore, the existing gradient-based training algorithms for RNNs, such as backpropagation through time (BPTT), can be employed for training the extended versions without any modifications.

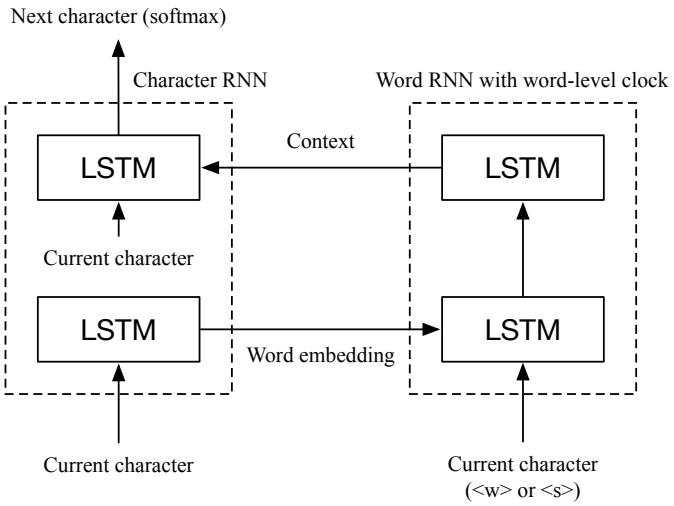
5.4 Character-Level Language Modeling with a Hierarchical RNN

The proposed hierarchical RNN (HRNN) architectures have several RNN modules with different clock rates as depicted in Figure 5.2. The higher level module employs a slower clock rate than the lower module, and the lower level module is reset at every clock of the higher level module. Specifically, if there are L hierarchy levels, then the RNN consists of L submodules. Each submodule l operates with an external clock $c_{l,t}$ and a reset signal $r_{l,t}$, where $l = 1, \cdots, L$. The lowest level module, $l = 1$, has the fastest clock rate, that is, $c_{1,t} = 1$ for all t. On the other hand, the higher level modules, $l > 1$, have slower clock rates and $c_{l,t}$ can be 1 only when $c_{l-1,t}$ is 1. Also, the lower level modules $l < L$ are reset by the higher level clock signals, that is, $r_{l,t} = c_{l+1,t}$.

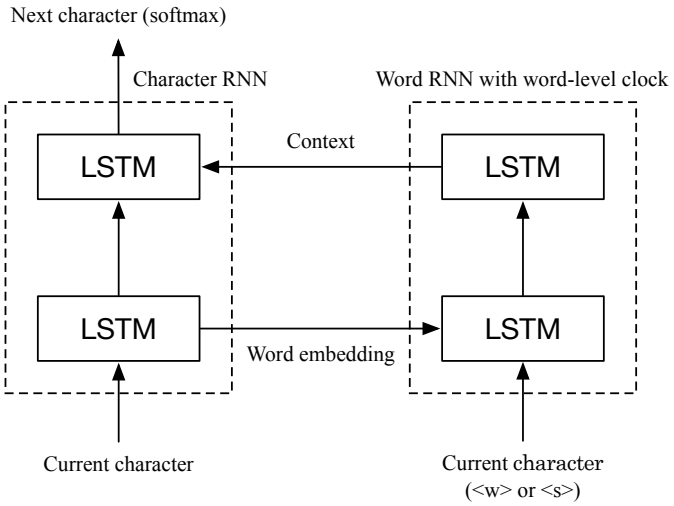
The hidden activations of a module, $l < L$, are fed to the next higher level module, $l+1$, *delayed by one time step* to avoid unwanted reset by $r_{l,t} = c_{l+1,t} = 1$. This hidden activation vector, or embedding vector, contains compressed short-term context information. The reset of the module by the higher level clock signals helps the module to concentrate on compressing only the short term information, rather than considering longer dependencies. The next higher level module, $l+1$, process this short-term information to generate the long-term context vector, which is fed back to the lower level module, l. There is no delay for this context propagation.

For character-level language modeling, we use a two-level ($L = 2$) HRNN with letting $l = 1$ be a character-level module and $l = 2$ be a word-level module. The word-level module is clocked at the word boundary input, `<w>`, which is usually a whitespace character. The input and softmax output layer is connected to the character-level module, and the current word boundary token (e.g. `<w>` or `<s>`) information is given to the word-level module. Since this HRNNs have a scalable architecture, we expect this HRNN CLM can be extended for modeling sentence-level contexts by adding an additional sentence-level module, $l = 3$. In this case, the sentence-level clock, $c_{3,t}$ becomes 1 when the input character is a sentence boundary token `<s>`. Also, the word-level module should be clocked at both the word boundary input, `<w>`, and the sentence boundary input, `<s>`. In this paper, the experiments are performed only with the two-level HRNN CLMs.

We propose two types of two-level HRNN CLM architectures. As shown in Figure 5.3, both models have two LSTM layers per submodule. Note that each connection has a weight matrix. In the HLSTM-A architecture, both LSTM layers in the character-level module receives one-hot encoded character input. Therefore, the second layer of the character-level module is a generative model conditioned by the



(a) HLSTM-A



(b) HLSTM-B

Figure 5.3: Two-level hierarchical LSTM (HLSTM) structures for CLMs.

context vector. On the other hand, in HLSTM-B, the second LSTM layer of the character-level module does not have direct connection from the character inputs. Instead, a word embedding from the first LSTM layer is fed to the second LSTM layer, which makes the first and second layers of the character-level module work together to estimate the next character probabilities when the context vector is given. The experimental results show that HLSTM-B is more efficient for CLM applications.

Since the character-level modules are reset by the word-boundary token (i.e. <w> or whitespace), the context vector from the word-level module is the only source for the inter-word context information. Therefore, the model is trained to generate the context vector that contains useful information about the probability distribution of the next word. From this perspective, the word-level module in both HRNN CLM architectures can be considered as a word-level RNN LM, where the input is a word embedding vector and the output is a compressed descriptor of the next word probabilities. Although the proposed model consists of several RNN modules with different timescales, these can be jointly trained by BPTT as described in Section 3.

5.5 Experiments

The proposed HRNN based CLMs are evaluated with two text datasets: the Wall Street Journal (WSJ) corpus [52] and One Billion Word Benchmark [76]. Also, we present an end-to-end speech recognition example, where HLSTM CLMs are employed for prefix tree-based beam search decoding.

The RNNs are trained with truncated backpropagation through time (BPTT) [30, 33]. Also, ADADELTA [59] and Nesterov momentum [60] is applied for weight up-

date. No regularization method, such as dropout [62], is employed. The training is accelerated using GPUs by training multiple sequences in parallel as described in Chapter 2.

5.5.1 Perplexity

In this section, our models are compared with other WLMs in the literature in terms of word-level perplexity (PPL). The word-level PPL of our models is directly converted from bits-per-character (BPC), which is the standard performance measure for CLMs, as follows:

$$PPL = 2^{BPC \times \frac{N_c}{N_w}} \tag{5.10}$$

where N_c and N_w are the number of characters and words in a test set, respectively. Note that sentence boundary symbols (<s>) are also regarded as characters and words.

5.5.1.1 Wall Street Journal (WSJ) Corpus

Dataset The Wall Street Journal (WSJ) corpus [52] is designed for training and benchmarking automatic speech recognition systems. For the perplexity experiments, we used the non-verbalized punctuation (NVP) version of the LM training data inside the corpus. The dataset consists of about 37 million words, where one percent of the total data is held out for the final evaluation and does not participate in training. All alphabets are converted to the uppercases.

Table 5.1: Perplexities of CLMs on the WSJ corpus

Model	Size	# Params	BPC	Word PPL
Deep LSTM	2x512	3.23 M	1.148	99.5
Deep LSTM	4x512	7.43 M	1.132	93.3
Deep LSTM	4x1024	29.54 M	1.101	82.4
HLSTM-A	4x512	7.50 M	1.089	78.5
HLSTM-B (no reset)	4x512	8.48 M	1.080	75.7
HLSTM-B	4x512	8.48 M	1.073	73.6
HLSTM-B	4x1024	33.74 M	1.058	69.2

Table 5.2: Perplexities of WLMs on the WSJ corpus in the literature

Model	# Params	PPL
KN 5-gram (no count cutoffs) [80]	-	80
RNN-640 + ME 4-gram feature [80]	2 G	59

Experimental results Table 5.1 shows the perplexities of traditional mono-clock deep LSTM and HLSTM based CLMs on the held-out set. Note that the size NxM means that the network consists of N LSTM layers, where each layer contains M memory cells. The HLSTM models show better perplexity performanes even when the number of LSTM cells or parameters is much smaller than that of the deep LSTM networks. Especially, HLSTM-B network with the size of 4x512 has about 9% lower perplexity than deep LSTM (4x1024) model, even with only 29% of parameters.

Importance of reset It is important to reset the character-level modules at the word-level clocks for helping the character-level modules to better concentrate on the short-term information. As observed in Table 5.1, removing the reset functionality of the character-level module of the HLSTM-B model results in degraded performance.

Table 5.3: Perplexities of the HRNN CLMs on the One Billion Word Benchmark

Model	Size	# Params	BPC	Word PPL
HLSTM-B	4x512	9.06 M	1.228	83.3
HLSTM-B	4x1024	34.90 M	1.140	60.7

Comparison with WLMs The non-ensemble perplexities of WLMs in the literature are presented in Table 5.2. The Kneser-Ney (KN) smoothed 5-gram model (KN-5) [81] is a strong non-neural WLM baseline. With the standard deep RNN based CLMs, it is very hard to beat KN-5 in terms of perplexity. However, it is surprising that all HLSTM models in Table 5.1 shows better perplexities than KN-5 does. The RNN based WLM model combined with the maximum entropy 4-gram feature [82, 80] shows much better results than the proposed HLSTM based CLM models. However, like most of the WLMs, it also needs a very large number (2 G) of parameters and cannot handle out-of-vocabulary (OOV) words.

5.5.1.2 One Billion Word Benchmark

Dataset The One Billion Word Benchmark [76] dataset contains about 0.8 billion words and roughly 800 thousand words of vocabulary. We followed the standard way of splitting the training and test data as in [76]. Each byte of UTF-8 encoded text is regarded as a character. Therefore, the size of the character set is 256.

Experimental results Due to the large amount of training data and weeks of training time, only two HLSTM-B experiments are conducted with the size of 4x512 and 4x1024. As shown in Table 5.3, there are large gap (22.5) in word-level perplexity between the two models. Therefore, further improvement in perplexity can be expected with bigger networks.

Table 5.4: Perplexities of WLMs on the One Billion Word Benchmark in the literature

Model	# Params	PPL
Sigmoid RNN-2048 [83]	4.1 G	68.3
Interpolated KN-5, 1.1B n-grams [76]	1.76 G	67.6
LightRNN [84]	**41 M**	**66**
Sparse non-negative matrix LM [85]	33 G	52.9
RNN-1024 + ME 9-gram feature [76]	20 G	51.3
CNN input + 2xLSTM-8192-1024 [72]	1.04 G	30.0

Comparison with WLMs The perplexities of other WLMs are summarized in Table 5.4. The proposed HLSTM-B model (4x1024) shows better perplexities than the interpolated KN-5 model with 1.1 billion n-grams [76] even though the number of parameters of our model is only 2% of that of the KN-5 model. Also, our model performs better than LightRNN [84], which is a word-level RNN LM that has about 17% more parameters than ours. However, much lower perplexities are reported with sparse non-negative matrix LM and the maximum entropy feature based RNN model [76], where the number of parameters are 33 G and 20 G, respectively. Recently, the state of the art perplexity of 30.0 was reported in [72] with a single model that has 1 G parameters. The model is basically a very large LSTM LM. However, a convolutional neural network (CNN) is used to generate word embedding of arbitrary character sequences as the input of the LSTM LM. Therefore, this model can handle OOV word inputs, however, still the model runs with a word-level clock.

5.5.2 End-to-End Automatic Speech Recognition (ASR)

In this section, we apply the proposed CLMs to the end-to-end automatic speech recognition (ASR) system to evaluate the models in more practical situation than just measuring perplexities. The CLMs are trained with WSJ LM training data as

Table 5.5: End-to-end ASR results on the WSJ Nov'92 20K evaluation set (`eval92`)

Model	Size	# Params	Word PPL	WER
Deep LSTM	4x512	7.43 M	93.3	8.36%
Deep LSTM	4x1024	29.54 M	82.4	7.85%
HLSTM-B	4x512	8.48 M	73.6	7.79%
HLSTM-B	4x1024	33.74 M	69.2	7.78%

in Section 5.5.1.1. Unlike WLMs, the proposed CLMs have very small number of parameters, so they can be employed for real-time character-level beam search.

The incremental speech recognition system proposed in Chapter 4 is used for the evaluation. The acoustic model is 4x512 unidirectional LSTM and end-to-end trained with the online connectionist temporal classification (CTC) loss proposed in Chapter 3. To train the acoustic model, the non-verbalized punctuation (NVP) portion of WSJ `SI-284` and `SI-ALL` training set is used as in Chapter 4. The acoustic features are 40-dimensional log-mel filterbank coefficients, energy and their delta and double-delta values, which are extracted every 10 ms with 25 ms Hamming window. The ASR models are evaluated on the WSJ Nov'92 20K evaluation set (`eval92`).

The results are summarized in Table 5.5, where the acoustic model is trained on `SI-284`. The beam-search decoding is performed on a prefix-tree with depth-pruning and width-pruning. The insertion bonus is 1.6, the LM weight is 2.0, and the beam width is 512. It is observed that the perplexity of LM and the word error rate (WER) have strong correlation. As shown in the table, we can achieve a better WER by replacing the traditional deep LSTM (4x1024) CLM with the proposed HLSTM-B (4x512) CLM, while reducing the number of LM parameters to 30%.

Further analysis is performed by redrawing Figure 4.5 with our best acoustic model trained on `SI-ALL` and two different CLMs: LSTM 2x512, which is used in

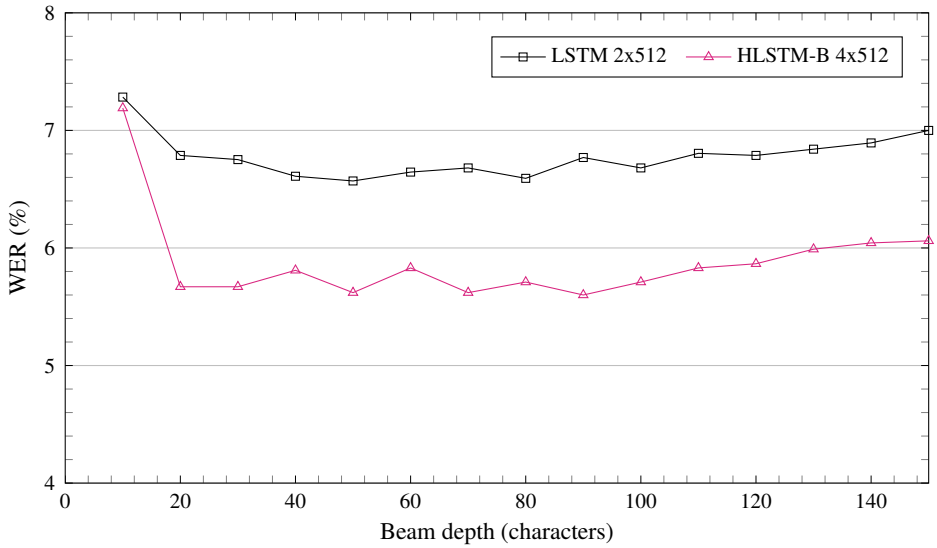


Figure 5.4: WER of the character-level ASR system with respect to the beam depth. Two different CLMs are used: LSTM 2x512 and HLSTM 4x512. The experiments are conducted with the acoustic RNN trained on `SI-ALL` and beam search is performed with the beam width of 512.

Chapter 4, and the proposed HLSTM-B 4x512. We used the fixed beam width of 512 and various beam depths from 10 to 150. As shown in Figure 5.4, the HLSTM based CLM leads to the reduction of WER roughly by 1% except when the beam depth is 10. This result is also consistent in Figure 5.5, which is drawn by fixing the beam depth to 50 and varying the beam width from 8 to 512. Note that the two CLMs have different number of parameters and this experiment is not intended for fair comparison of the two different RNN architectures.

5.6 Concluding Remarks

In this paper, hierarchical RNN (HRNN) based CLMs are proposed. The HRNN consists of several submodules with different clock rates. Therefore, it is capable

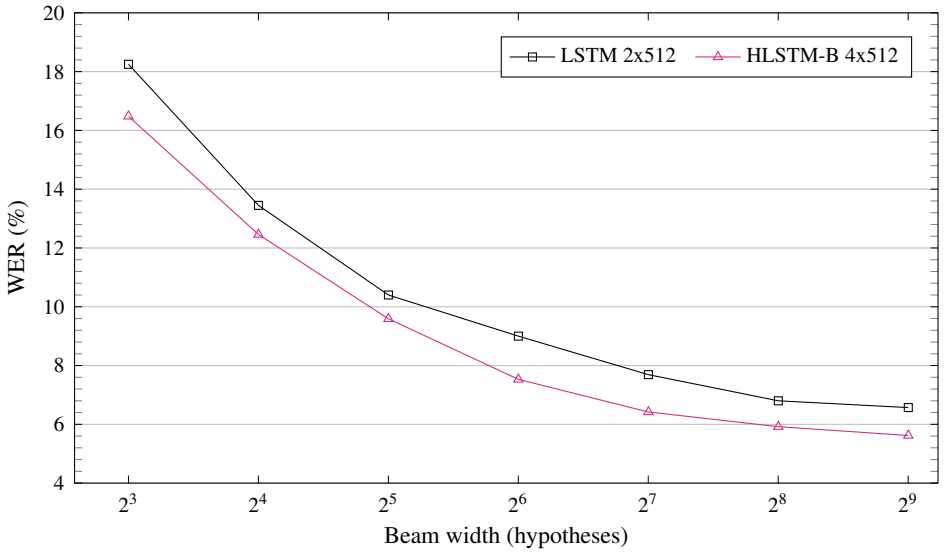


Figure 5.5: WER of the character-level ASR system with respect to the beam width. Two different CLMs are used: LSTM 2x512 and HLSTM 4x512. The experiments are conducted with the acoustic RNN trained on `SI-ALL` and beam search is performed with the beam depth of 50.

of learning long-term dependencies as well as short-term details. The experimental results on One Billion Benchmark show that HLSTM-B networks significantly outperform Kneser-Ney 5-gram LMs with only 2% of parameters. Although other RNN-based WLMs show better performance than our models, they have impractically many parameters. On the other hand, as shown in the WSJ speech recognition example, the proposed model can be employed for the real-time speech recognition with less than 10 million parameters. Also, CLMs can handle OOV words by nature, which is a great advantage for the end-to-end speech recognition and many NLP tasks. One of the interesting future work is to train the clock signals, instead of using manually designed ones. Also, it would be interesting to see how this hierarchical architecture will perform when the level of hierarchy increases.

Chapter 6

Conclusion

In this dissertation, we have discussed the design of RNN-based end-to-end online ASR, efficient training algorithm and system for it, and the improved character-level RNN LM for performance enhancement.

Specifically, a flexible and efficient GPU-based training system for RNN is described in Chapter 2. This training system is equipped with truncated BPTT algorithm and capable of training unidirectional RNNs with very long sequences for preventing the RNN explosion during online decoding. RNNs are represented by directed graph structures, and automatically parallelized for GPU-based training. The system was used for training all the RNNs in this dissertation. Especially, the RNN LMs were trained with this system without further modification.

In Chapter 3, the CTC forward-backward algorithm is also modified and integrated with the truncated BPTT algorithm and the training system in the previous chapter. As a result, the trained RNN AM can decode infinitely long input speech without any RNN explosion problem. Moreover, the training time is significantly re-

duced due to increased parallelism.

The proposed character-level online ASR model is introduced in Chapter 4. The model employs a character-level RNN AM, which is trained with CTC loss, and a character-level RNN LM for augmenting the speech recognition results. The best hypothesis is found by prefix-tree based beam search. To prevent monotonic growth of the decoding tree, "depth" pruning is applied as well as conventional "width" pruning. The proposed character-level online ASR system does not require lexicon, naturally handles OOV words, and is capable of decoding infinitely long stream of input speech.

In Chapter 5, the character-level LM is improved with hierarchical RNN structures and applied for the proposed online ASR system. As a result, the amount of computation is significantly reduced while achieving improved speech recognition accuracy. Also, it has been shown that this character-level LM with hierarchical RNN architecture outperforms a lightweight word-level RNN LM with a comparable amount of parameters.

Bibliography

[1] L. Rabiner and B.-H. Juang, *Fundamentals of speech recognition*. Prentice Hall, 1993.

[2] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[5] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[6] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[7] S. Goldwater, D. Jurafsky, and C. D. Manning, "Which words are hard to recognize? prosodic, lexical, and disfluency factors that increase speech recognition error rates," *Speech Communication*, vol. 52, no. 3, pp. 181–200, 2010.

[8] A. Cutler, D. Dahan, and W. Van Donselaar, "Prosody in the comprehension of spoken language: A literature review," *Language and Speech*, vol. 40, no. 2, pp. 141–201, 1997.

[9] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

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

[11] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, *et al.*, "Deep Speech 2: End-to-end speech recognition in english and mandarin," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016, pp. 173–182.

[12] G. Sagerer, H. Rautenstrauch, G. A. Fink, B. Hildebrandt, A. Jusek, and F. Kummert, "Incremental generation of word graphs," in *Proceedings of the 4th International Conference on Spoken Language (ICSLP)*, vol. 4. IEEE, 1996, pp. 2143–2146.

[13] E. O. Selfridge, I. Arizmendi, P. A. Heeman, and J. D. Williams, "Stability and accuracy in incremental speech recognition," in *Proceedings of the SIGDIAL 2011 Conference*. Association for Computational Linguistics, 2011, pp. 110–119.

[14] I. McGraw and A. Gruenstein, "Estimating word-stability during incremental speech recognition," *Training*, vol. 17, no. 27,327, pp. 6–4, 2011.

[15] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[16] D. Krueger and R. Memisevic, "Regularizing RNNs by stabilizing activations," in *4th International Conference on Learning Representations*, 2016.

[17] A. Zeyer, R. Schlüter, and H. Ney, "Towards online-recognition with deep bidirectional lstm acoustic models," *Proceedings of INTERSPEECH 2016*, 2016.

[18] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*. ACM, 2006, pp. 369–376.

[19] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011, pp. 1017–1024.

[20] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT 2010*. Springer, 2010, pp. 177–186.

[21] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin, "A fast parallel stochastic gradient method for matrix factorization in shared memory systems," *Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 1, p. 2, 2015.

[22] K. Hwang and W. Sung, "Single stream parallelization of generalized LSTM-like RNNs on a GPU," in *Proceedings of 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015.

[23] ——, "Sequence to sequence training of CTC-RNNs with partial windowing," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016, pp. 2178–2187.

[24] ——, "Character-level incremental speech recognition with recurrent neural networks," in *Proceedings of 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016.

[25] ——, "Character-level language modeling with hierarchical recurrent neural networks," in *Proceedings of 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017.

[26] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Proceedings of 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 5528–5531.

[27] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[28] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.

[29] D. Monner and J. A. Reggia, "A generalized LSTM-like training algorithm for second-order recurrent neural networks," *Neural Networks*, vol. 25, pp. 70–83, 2012.

[30] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[31] X. Chen, Y. Wang, X. Liu, M. J. Gales, and P. C. Woodland, "Efficient GPU-based training of recurrent neural network language models using spliced sentence bunch," in *Proceedings of INTERSPEECH 2014*, 2014.

[32] R. Tarjan, "Depth-first search and linear graph algorithms," *Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[33] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation*, vol. 2, no. 4, pp. 490–501, 1990.

[34] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, "Sample size selection in optimization methods for machine learning," *Mathematical Programming*, vol. 134, no. 1, pp. 127–155, 2012.

[35] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *5th International Conference on Learning Representations*, 2015.

[36] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 3104–3112.

[37] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[38] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *Proceedings of 2007 International Conference on Artificial Neural Networks (ICANN)*. Springer, 2007, pp. 220–229.

[39] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 6645–6649.

[40] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, *et al.*, "Deep Speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[41] D. Bahdanau, J. Chorowski, D. Serdyuk, Y. Bengio, *et al.*, "End-to-end attention-based large vocabulary speech recognition," in *Proceedings of 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 4945–4949.

[42] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Proceedings of 2015 Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 577–585.

[43] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proceedings of 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 4960–4964.

[44] A. L. Maas, Z. Xie, D. Jurafsky, and A. Y. Ng, "Lexicon-free conversational speech recognition with neural networks," in *Proceedings of 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015.

[45] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Proceedings of 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 167–174.

[46] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Proceedings of 2008 Advances in Neural Information Processing Systems (NIPS)*, 2008, pp. 577–584.

[47] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke, "A novel word spotting method based on recurrent neural networks," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 211–224, 2012.

[48] G. Fink, C. Schillo, F. Kummert, G. Sagerer, *et al.*, "Incremental speech recognition for multimodal interfaces," in *Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, vol. 4. IEEE, 1998, pp. 2012–2017.

[49] A. Graves *et al.*, *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, vol. 385.

[50] K. Hwang, M. Lee, and W. Sung, "Online keyword spotting with a character-level recurrent neural network," *arXiv preprint arXiv:1512.08903*, 2015.

[51] D. B. Fry, "Simple reaction-times to speech and non-speech stimuli," *Cortex*, vol. 11, no. 4, pp. 355–360, 1975.

[52] D. B. Paul and J. M. Baker, "The design for the Wall Street Journal-based CSR corpus," in *Proceedings of the Workshop on Speech and Natural Language*. Association for Computational Linguistics, 1992, pp. 357–362.

[53] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1," *NASA STI/Recon Technical Report N*, vol. 93, p. 27403, 1993.

[54] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Springer, 1990, pp. 227–236.

[55] J. Sohn, N. S. Kim, and W. Sung, "A statistical model-based voice activity detection," *Signal Processing Letters*, vol. 6, no. 1, pp. 1–3, 1999.

[56] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[57] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *The Journal of Machine Learning Research (JMLR)*, vol. 3, pp. 115–143, 2003.

[58] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, *et al.*, *The HTK book*. Entropic Cambridge Research Laboratory Cambridge, 1997, vol. 2.

[59] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[60] Y. Nesterov, "A method of solving a convex programming problem with convergence rate O (1/k2)," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[61] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 8624–8628.

[62] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[63] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[64] K.-F. Lee and H.-W. Hon, "Speaker-independent phone recognition using hidden Markov models," *Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 11, pp. 1641–1648, 1989.

[65] M. Killer, S. Stüker, and T. Schultz, "Grapheme based speech recognition." in *Proceedings of INTERSPEECH 2003*, 2003.

[66] M. Bisani and H. Ney, "Open vocabulary speech recognition with flat hybrid models." in *Proceedings of INTERSPEECH 2005*, 2005, pp. 725–728.

[67] A. Y. Hannun, A. L. Maas, D. Jurafsky, and A. Y. Ng, "First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs," *arXiv preprint arXiv:1408.2873*, 2014.

[68] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Proceedings of 2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2013, pp. 273–278.

[69] P. C. Woodland, J. J. Odell, V. Valtchev, and S. J. Young, "Large vocabulary continuous speech recognition using HTK," in *Proceedings of 1994 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2. IEEE, 1994, pp. II–125.

[70] P. F. Brown, J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin, "A statistical approach to machine translation," *Computational Linguistics*, vol. 16, no. 2, pp. 79–85, 1990.

[71] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Proceedings of INTERSPEECH 2010*, vol. 2, 2010, p. 3.

[72] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.

[73] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Proceedings of 2013 Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 190–198.

[74] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," in *Proceedings of 30th AAAI Conference on Artificial Intelligence*, 2016.

[75] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[76] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," *arXiv preprint arXiv:1312.3005*, 2013.

[77] W. Ling, T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso, "Finding function in form: Compositional character models for open vocabulary word representation," in *Proceedings of 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1520–1530.

[78] Y. Miyamoto and K. Cho, "Gated word-character recurrent language model," in *Proceedings of 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1992–1997.

[79] W. Ling, I. Trancoso, C. Dyer, and A. W. Black, "Character-based neural machine translation," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, vol. 357–361, 2016.

[80] T. Mikolov, "Statistical language models based on neural networks," Ph.D. dissertation, Brno University of Technology, 2012.

[81] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *Proceedings of 1995 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1. IEEE, 1995, pp. 181–184.

[82] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," in *Proceedings of 2012 IEEE Spoken Language Technology Workshop*. IEEE, 2012, pp. 234–239.

[83] S. Ji, S. Vishwanathan, N. Satish, M. J. Anderson, and P. Dubey, "BlackOut: Speeding up recurrent neural network language models with very large vocabularies," in *4th International Conference on Learning Representations*, 2016.

[84] X. Li, T. Qin, J. Yang, X. Hu, and T. Liu, "LightRNN: Memory and computation-efficient recurrent neural networks," in *Proceedings of 2016 Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 4385–4393.

[85] N. Shazeer, J. Pelemans, and C. Chelba, “Sparse non-negative matrix language modeling for skip-grams,” in *Proceedings of INTERSPEECH 2015*, 2015, pp. 1428–1432.

국문 초록

재귀형 인공신경망(recurrent neural network, RNN)은 최근 시퀀스-투-시퀀스(sequence-to-sequence) 방식의 여러 모델에서 좋은 성능을 보여 왔다. 최근의 음성인식에서 사용하는 종단간(end-to-end) 훈련 방식의 발전으로 인해, RNN은 일련의 오디오 특징(feature)을 입력으로 하고 일련의 글자(character) 혹은 단어들을 출력으로 하는 단일한 함수를 학습할 수 있게 되었다. 이 함수는 중간에 음소 단위 혹은 발음 사전(lexicon) 단위의 변환을 거치지 않는다. 지금까지, 대부분의 종단간 음성인식은 기존 방식으로 얻은 높은 정확도를 따라가는 데 초점이 맞춰져 있었다. 하지만, 비록 종단간 음성인식 모델이 기존 음성인식 모델만큼의 정확도를 달성했음에도, 이 모델은 보통 미리 잘라진 오디오 데이터를 사용하는 발화 단위의 음성인식에 사용되었고, 실시간으로 연속적인 오디오 데이터를 받아 사용하는 음성인식에는 잘 사용되지 않았다. 이것은 미리 잘라진 데이터로 학습한 RNN은 매우 긴 오디오 입력에 대해서도 잘 동작하도록 일반화(generalization)가 되기 어려웠기 때문이다.

위 문제를 해결하기 위해, 본 논문에서는 무한히 긴 시퀀스를 사용하는 RNN 훈련 방법을 제안한다. 먼저, 이를 위한 효과적인 그래픽 프로세서(graphics processing unit, GPU) 기반 RNN 훈련 프레임워크(framework)를 설명한다. 이 프레임워크는 제한된 시간축 역전파(truncated backpropagation through time, truncated BPTT)를 사용해 훈련되며, 덕분에 실시간으로 들어오는 연속적인 데이터를 사용하여 훈련할 수 있다. 다음으로, 연결성 시계열 분류기(connectionist temporal classification, CTC) 알고리즘의 손실(loss) 계산 방식을 변형한 실시간 CTC 학습

알고리즘을 선보인다. 새롭게 선보인 CTC 손실 계산 알고리즘은 truncated BPTT 기반의 RNN 훈련에 바로 적용될 수 있다.

다음으로, RNN만으로 구성된 종단간 실시간 음성인식 모델을 소개한다. 이 모델은 크게 CTC 출력을 사용하는 음향(acoustic) RNN과 글자 단위 RNN 언어 모델(language model)로 구성된다. 그리고, 접두사 트리(prefix-tree) 기반의 새로운 빔 탐색(beam search)이 사용되어 무한한 입력 오디오에 대해 디코딩(decoding)을 수행할 수 있다. 이 디코딩 방식에는 새로운 빔 가지치기(beam pruning) 알고리즘이 도입되어 트리 구조의 크기가 지수적으로 증가하는 것을 방지한다. 위 음성인식 모델에는 별도의 음소 모델이나 발음 사전이 포함되어 있지 않고, 무한히 긴 일련의 오디오에 대해 디코딩을 수행할 수 있다는 특징이 있다. 위 모델은 또한 다른 종단간 모델에 비해 매우 적은 메모리를 사용하면서도 비견될 만한 정확도를 보인다.

마지막으로, 본 논문에서는 계층형 구조(hierarchical structure)를 이용해 글자 단위 RNN 언어 모델의 성능을 향상시켰다. 특히, 이 글자 단위 RNN 모델은 비슷한 파라미터 수를 갖는 단어 단위 RNN 언어 모델보다 개선된 예측 복잡도(perplexity)를 달성하였다. 또한, 이 글자 단위 RNN 언어 모델을 앞서 설명한 글자 단위 실시간 음성인식 시스템에 적용하여 더욱 적은 연산을 사용하면서도 음성인식 정확도를 향상시킬 수 있었다.

주요어 : 음성인식, 재귀형 인공 신경망, 종단간 학습, 실시간 추론
학번 : 2010-23300