Ph.D. DISSERTATION

End-to-End Neural Network-based Speech Recognition for Mobile and Embedded Devices

모바일 및 임베디드 환경을 위한
인공 신경망 기반 음성 인식

BY

JINHWAN PARK

AUGUST 2020

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

Ph.D. DISSERTATION

End-to-End Neural Network-based Speech Recognition for Mobile and Embedded Devices

모바일 및 임베디드 환경을 위한
인공 신경망 기반 음성 인식

BY

JINHWAN PARK

AUGUST 2020

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

End-to-End Neural Network-based Speech Recognition for Mobile and Embedded Devices

모바일 및 임베디드 환경을 위한
인공 신경망 기반 음성 인식

지도교수 성 원 용
이 논문을 공학박사 학위논문으로 제출함

2020년 8월

서울대학교 대학원

전기 정보 공학부

박 진 환

박진환의 공학박사 학위 논문을 인준함

2020년 8월

Abstract

Real-time automatic speech recognition (ASR) on mobile and embedded devices has been of great interest in recent years. Deep neural network-based automatic speech recognition demands a large number of computations, while the memory bandwidth and power storage of mobile devices are limited. The server-based implementation is often employed, but this increases latency or privacy concerns. Therefore, the need of the on-device ASR system is increasing. Recurrent neural networks (RNNs) are often used for the ASR model. The RNN implementation on embedded devices can suffer from excessive DRAM accesses, because the parameter size of a neural network usually exceeds that of the cache memory. Also, the parameters of RNN cannot be reused for multiple time-steps due to its feedback structure. To solve this problem, multi-time step parallelizable models are applied for speech recognition. The multi-time step parallelization approach computes multiple output samples at a time with the parameters fetched from the DRAM. Since the number of DRAM accesses can be reduced in proportion to the number of parallelization steps, a high processing speed can be achieved for the parallelizable model.

In this thesis, a connectionist temporal classification (CTC) model is constructed by combining simple recurrent units (SRUs) and depth-wise 1-dimensional convolution layers for multi-time step parallelization. Both the character and word piece models are developed for the CTC model, and the corresponding RNN based language models are used for beam search decoding. A competitive WER for WSJ corpus is achieved using the entire model size of approximately 15MB. The system operates in real-time speed using only a single core ARM without GPU or special hardware.

A low-latency on-device speech recognition system with a simple gated convolutional network (SGCN) is also proposed. The SGCN shows a competitive recognition accuracy even with 1M parameters. 8-bit quantization is applied to reduce the memory size and computation time. The proposed system features an online recognition with a 0.4s latency limit and operates in 0.2 RTF with only a single 900MHz CPU core.

In addition, an attention-based model with the depthwise convolutional encoder is proposed. Convolutional encoders enable faster training and inference of attention models than recurrent neural network-based ones. However, convolutional models often require a very large receptive field to achieve high recognition accuracy, which not only increases the parameter size but also the computational cost and run-time memory footprint. A convolutional encoder with a short receptive field length often suffers from looping or skipping problems. We believe that this is due to the time-invariance of convolutions. We attempt to remedy this issue by adding positional information to the convolution-based encoder. It is shown that the word error rate (WER) of a convolutional encoder with a short receptive field size can be reduced significantly by augmenting it with positional information. Visualization results are presented to demonstrate the effectiveness of incorporating positional information. The streaming end-to-end ASR model is also developed by applying monotonic chunkwise attention.

keywords: Neural networks, Streaming speech recognition, On-device machine learning

student number: 2015-22786

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 End-to-End Automatic Speech Recognition with Neural Networks

Deep neural network-based models have greatly improved the accuracy of automatic speech recognition (ASR) tasks with end-to-end training. Unlike traditional ASR, which requires the complicated system structure and training processes, the end-to-end ASR system directly learns the output transcriptions from the input speech. The end-to-end models include connectionist temporal classification (CTC) [1, 2], attention-based models [3, 4], and transducers [5, 6, 7]. The end-to-end neural network-based ASR models have achieved a state-of-the-art accuracy on various large-scale datasets [8].

Neural network-based ASR is usually based on recurrent neural networks (RNNs), such as long short-term memory (LSTM) [9] or gated recurrent unit (GRU) [10], which are capable of learning the long-term context of input sequences. The RNN-based systems demand a huge amount of multiply and add operations, and often contain more than millions of parameters. This requires excessive memory access and high computing power. To fix this, model compression techniques are actively studied for the efficient implementation of the ASR system. The compression algorithms include quantization [11, 12, 13], pruning [14, 15, 16], and matrix decomposition [17]. However, these techniques require additional data rearrangement or dedicated hardware support [18] to obtain improvement in throughput. Therefore, commercial services in the real environment usually adopt cloud-based implementations with high-performance GPU servers.

1.2 Challenges on On-device Implementation of Neural Network-based ASR

The shortcomings of server-based ASR implementations include the network latency and privacy concerns resulting from the transfer of users' data to the server. The server operating cost is also a problem for the service provider. Consequently, there is a huge demand for an on-device ASR implementation for the internet of things (IoT) and mobile devices to overcome the problems of the cloud-based systems.

The memory access overhead often becomes the main bottleneck of on-device neural network-based ASR. Typically, neural network-based ASR requires 1 to 10 Giga arithmetic operations per second. This range of computational cost is affordable by single-core CPUs when optimized with single instruction multiple data (SIMD) operations. However, memory accesses are still major issues. Usually, the parameter size of the speech recognition model is larger than 100 MB, which is far more than the available cache size of CPUs. When computing RNN, the weights should be fetched from the main memory to cache every input frame since it requires the previous time-step output for the current computation. Although a large number of memory accesses can be hidden by multi-stream parallel processing in server-based implementation, this approach is not feasible for on-device implementations where usually targets a single user.

1.3 Parallelizable Neural Network Architecture

To avoid memory access issues of RNNs, the parallelizable models are often employed instead. Conventional RNNs such as an LSTM [9] have a feedback structure on the matrix-vector multiplication, which must be computed serially for the single stream. Since matrix-vector multiplication is the most computation-intensive operation, the computation of a single sequence is inefficient for the RNN. Recent studies have proposed the simplified RNN structures which have feedback on element-wise computations only. The simplified RNNs include strongly-typed recurrent unit [19], Quasi-RNN [20], simple recurrent unit (SRU) [21], and gated impulse linear recurrent (GILR) unit [22]. These RNNs have shown a solid performance for natural language processing, such as language modeling or machine translation. They also have a far faster training and inference speed when compared to LSTM or GRU. However, these models have a worse accuracy than conventional RNNs. We consider that the simplified recurrence is inadequate to learn the long-term context required for the speech recognition task.

Besides recurrent models, fully convolutional neural networks have also been used for sequence learning. They achieved a good performance on machine translation [23, 24, 25], language modeling [26], and speech recognition [27]. More recently, self-attention based models [28] have been proposed and widely been used for sequential tasks. These models have also achieved for the end-to-end speech recognition task. However, these models require a large size of the receptive field for learning the long-term context for the speech recognition task. This increases the computational overhead and memory footprint for the convolution and self-attention-based models, which makes it difficult for these models to be used in memory-constrained devices.

1.4 Scope of Dissertation

As discussed in previous sections, there are a lot of challenges in developing the on-device ASR system, which mainly comes from the inefficient utilization of cache. In this study, a method to develop memory-efficient on-device ASR using a parallelizable neural network structure is proposed.

This dissertation is organized as follows. In Chapter 2, a simple recurrent unit (SRU) is applied to CTC-based speech recognition. As discussed in Section 1.3, SRU has low recognition performance when applied to the end-to-end speech recognition. SRU is considered to have a lack of capacity in learning the long-term context. To compensate for simplified recurrence, depthwise convolutions are used to increase the receptive field of SRU. It is shown that SRU with 1-D depthwise convolution has lower WER than unidirectional LSTM. RNN-based language models are applied for beam search decoding. A competitive WER for WSJ corpus is achieved using the entire model size of around 15MB. The system operates in real-time speech using only a single core ARM without GPU or special hardware.

In Chapter 3, a 1MB memory footprint ASR model is developed, which could be applied for always-on speech recognizer or keyword spotting. To reduce the memory size, a depthwise convolution-based structure is used. The model shows a competitive recognition accuracy even with 1M parameters. 8-bit quantization is also applied to further reduce the memory footprint and computation time. The proposed system features the online recognition fulfilling the 0.4s latency limit, operating with the real-time factor of 0.2 by utilizing only a single 900MHz CPU core.

The attention-based end-to-end speech recognition models usually have lower WER than CTC-based models when the external LM is not available. In Chapter 4, a method to train attention-based models with convolutional encoders is proposed. Convolutional models often require a very large receptive field to achieve high recognition accuracy, which not only increases the parameter size but also the computational cost and run-time memory footprint. A convolutional encoder with a short receptive field length can suffer from looping or skipping problems when the input utterance contains the same words as nearby sentences. We believe that this is due to the insufficiency of positional information. We try to remedy this problem by adding positional information to the convolution-based encoder. It is shown that WER of attention-based models with a convolutional encoder is reduced by augmenting it with positional encoding.

The conventional attention-based models are difficult to use for streaming speech recognition since it needs the entire input sequence to be processed before generating the first output label. To overcome this problem, local attention, such as the monotonic chunkwise attention [29] has been proposed. In Chapter 5, the convolutional encoder is applied to the monotonic chunkwise attention-based model. The receptive field size for convolutional encoder is reduced with the method proposed in Chapter 4, which is crucial for reducing the working memory for streaming speech recognition.

The materials of Chapter 2 and 3 were previously published by the author in [30, 31], respectively. In addition, Chapter 4 and 5 have been submitted to InterSpeech 2020 and Advances in Neural Information Processing Systems (NeurIPS 2020).

Chapter 2

Simple Recurrent Units for CTC-based End-to-End Speech Recognition

2.1 Introduction

Recently, neural network technology has greatly improved the accuracy of automatic speech recognition (ASR), and many applications are being developed for smartphones and intelligent personal assistants. Many researches on end-to-end speech recognition are being conducted to replace the hidden Markov model (HMM) based technique which has been used for many years. The end-to-end models with neural networks include connectionist temporal classification (CTC)-trained recurrent neural networks (RNN) [1, 2], encoder-decoder architectures [4, 3], and RNN transducers [5, 6]. Although HMM-based algorithms can be considered arithmetically efficient, they demand many irregular memory accesses and a large memory foot-print usually exceeding a few hundred MBs. In contrast, RNN based ASR has the advantage of low memory footprint; however, it demands numerous arithmetic operations for real-time inference. Consequently, server-based ASR implementations are mostly used for real services, which however has the problem of response delay and user privacy issues. Therefore, there is a huge demand for on-device ASR implementation not only for smartphones but also for many internet of things (IoT) devices [32].

There have been many researches on accelerating the inference of neural networks by employing special-purpose hardware or GPUs. Our estimate of a fully neural network based single user speech recognition demands about 1 Giga arithmetic operations per second, which is not a formidable barrier because a single instruction multiple data (SIMD) instruction can conduct four to eight arithmetic operations at a time and the clock frequency of a CPU is around 1 GHz. The real problem is the cache misses because the model size of RNN, which is over 10 MB for most ASR, is usually much larger than the cache size of embedded CPUs. For example, ARM Cortex-A57 has a 2 MB L2 cache at most. Due to the sequential nature of RNN, the parameters should be fetched from the DRAM at each time step, which implies continuous cache misses. In GPU or server-based implementations, this problem is hidden by batch or multi-stream parallel processing.

In embedded devices, however, only one stream is executed, because it usually targets a single user. To solve the memory access problem, we apply multi-time step parallel processing for which multiple consecutive frames are computed concurrently, and the number of DRAM accesses can be reduced in proportion to the number of parallelization steps. Unfortunately, the multi-time step parallelization cannot be applied to popular RNN structures, such as long short-term memory (LSTM) [9] or gated recurrent unit (GRU) [10], because they contain complex input-output dependencies. Recent studies demonstrated linear RNNs with simplified feedback, which can be used for multi-time step parallelization [20, 21, 22, 19]. Quasi RNN (QRNN) and simple recurrent unit (SRU) are a kind of linear RNNs. However, when applied to acoustic modeling, the accuracy with linear RNN was not as good as that of LSTM RNN. We combined depth-wise 1-dimensional (1-D) convolution with linear RNN and obtained very good accuracy exceeding that of LSTM with a comparable parameter size.

The developed speech recognition system employs RNN based language models (LMs) instead of n-gram based ones. Character and word piece based LMs are used for beam search decoding. In addition, we also try hierarchical character LM (HCLM) for improved performance, and by which we can achieve an accuracy comparable to Deep Speech 2 [2] for Wall Street Journal (WSJ) corpus, with 10 times less parameters. The RNN LMs are based on LSTM or GRU because multiple streams are executed concurrently for beam search decoding. We reduce the overhead of DRAM accesses by executing multiple streams of RNN LMs at a time, where the stream size depends on the beam search width. For efficient beam search decoding, we early prune the output symbols of low probability in the acoustic model (AM).

The implementation operates in real-time on the ARM Cortex-A57 based embedded system without GPU support. The model size of the proposed speech recognition system including CTC-AM and RNN LM is about 15 MB with 8-bit parameters which is far smaller than that of conventional HMM-based systems.

This chapter is organized as follows. In Section 2.2, we review the related works and recently proposed RNNs. We introduce the proposed speech recognition system in Section 2.3. The experimental results including the execution time analysis are shown in Section 2.4. Section 2.5 concludes this chapter.

2.2 Related Works

Most mobile speech recognition methods have relied on WFST (weighted finite state transducer) based algorithms mainly because of their low arithmetic requirements [33]. However, a WFST network usually demands a foot-print of more than a few hundred MB because of the integrated n-gram based LM. Scattered and unaligned memory accesses also hinder efficient implementation of WFST networks.

Recently, fully neural network based speech recognition, which combines RNN based AM and LM, has drawn considerable attention. For efficient implementation of RNN, several model compression techniques have been developed, such as pruning [14], quantization [11], and matrix decomposition [17]. However, these techniques need efficient data rearrangement and decompression; therefore, they are not explored in this work. However, they can be combined with the proposed method. We applied 8-bit quantization to further reduce the execution time and the model size.

Recently, quasi RNN (QRNN) and simple recurrent unit (SRU) were developed for the purpose of fast training and inference of very long sequences [20, 21]. These RNNs only employ simple feedback that can be represented as linear recurrence equations, and allow not only fast training but also multi-time step parallel inference. QRNN showed a good performance comparable to LSTM in language modeling and machine translation.

Convolutional neural networks (CNNs) have also been used for sequence learning, such as machine translation [23, 24], language modeling [26], and speech recognition [27]. CNN has the advantage of parallel processing because there is no dependency between the input and output. However, CNN usually requires a large amount of feature maps, while RNN only needs to store the current cell and output state vectors in each layer. In this sense, we consider that RNN is more advantageous compared to CNN for on-device inference if multi-time step parallelization can be applied for memory access reduction.

2.3 Speech Recognition Algorithm

The target speech recognition system in this work consists of CTC-trained AM, RNN LM, and beam search decoder. In particular, we have developed two models; one is based on character as the output units of AM and LM, and the other is based on word piece units [34]. The word piece model is advantageous for real-time execution because the AM can operate at a low frame rate. In this section, we describe each component of the system in detail.

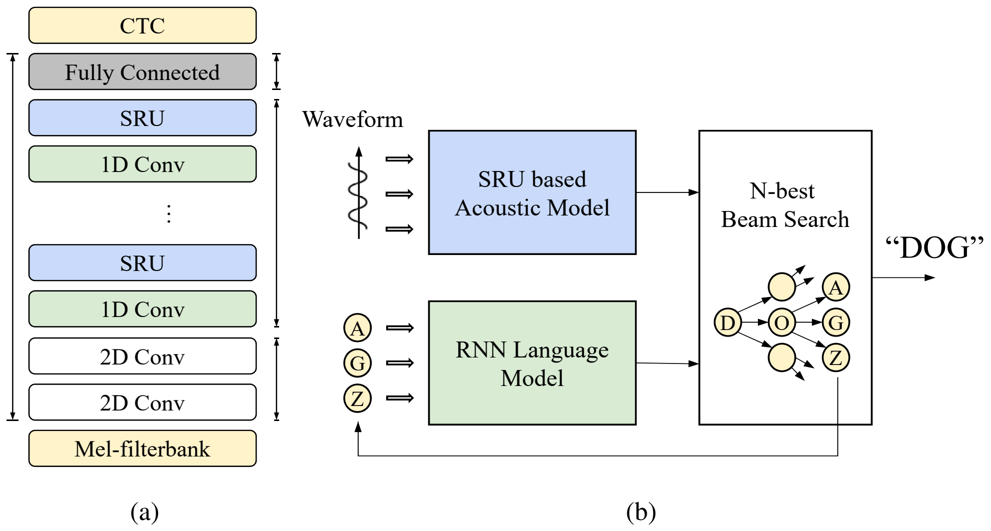


Figure 2.1: (a) The architecture of the neural network model used for acoustic modeling. (b) The system consists of RNN AM, RNN LM, and beam search decoding.

2.3.1 Acoustic modeling

The proposed AM architecture is shown in Figure 2.1 (a); it is composed of two 2-D convolutional layers, six recurrent layers, and the final fully connected layer. The output labels are either graphemes (characters) or word pieces. The AM RNN is trained with CTC loss [35]. The input of the convolutional layer consists of three 2-D feature maps with the time and frequency axes, and each feature map is formed with the mel-filter bank output, its delta, or double-delta. We employed two 2-D convolutional layers with a filter size of 5, as proposed in [2, 36]. The 2-D convolutional layers not only improve the recognition performance but also down-sample the input frames by two. Thus, the down-sampling in the convolutional layers not only reduces the computational complexity of the recurrent layers in AM but also simplifies the decoding.

The developed model contains six recurrent layers, and each layer consists of 1-D convolution unit and SRU. The recurrent layers adopt SRU which only employs cell-state feedback [21]. Given an input vector $\mathbf{x}_t \in \mathbb{R}^{d_{in}}$, the output $\mathbf{h}_t \in \mathbb{R}^N$ and the cell state $\mathbf{c}_t \in \mathbb{R}^N$ is computed in SRU as depicted in Eq. (1). Inspired from *ifo-pooling* of

QRNN, we replaced $(1 - \mathbf{f}_t)$ to input gate $\mathbf{i}_t$ [20]. We refer this as i-SRU in this work. In our experiments, adding the input gate not only improves the robustness in training of SRU but also leads to a lower error rate. We also explored the location of $\tanh$ in our preliminary experiments, and found that locating $\tanh$ as shown in Eq. (2.2) yielded slightly better results.

$$\begin{aligned}
&\text{SRU:}\\
&\hat{\mathbf{x}}_t = \mathbf{W}_z \mathbf{x}_t + \mathbf{b}_z,\\
&\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{b}_f),\\
&\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{b}_o),\\
&\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \hat{\mathbf{x}}_t,\\
&\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) + (1 - \mathbf{o}_t) \odot \mathbf{x}_t
\end{aligned} \tag{2.1}$$

$$\begin{aligned}
&\text{i-SRU:}\\
&\hat{\mathbf{x}}_t = \tanh(\mathbf{W}_z \mathbf{x}_t + \mathbf{b}_z),\\
&\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{b}_f),\\
&\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{b}_i),\\
&\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{b}_o),\\
&\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{x}}_t\\
&\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t + (1 - \mathbf{o}_t) \odot \mathbf{x}_t
\end{aligned} \tag{2.2}$$

where $\mathbf{W}_z, \mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_o \in \mathbb{R}^{N \times d_{in}}$ and $\mathbf{b}_z, \mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_o \in \mathbb{R}^N$ are trainable parameters. When the i-SRU is trained alone, the resulting WER is significantly worse than that of LSTM. To overcome this problem, we add a depth-wise 1-D convolutional layer at the input of each recurrent layer. This needs $O(k \times d_{in})$ additional parameters when the filter width of k is used for the convolution. The number of parameters used in 1-D convolutional layers is much smaller than that for recurrent layers, which is

$O(N \times d_{in})$. This is similar to QRNN with a filter size of $k > 1$. However, QRNN needs $O(k \times N \times d_{in})$ parameters. We obtained significant performance improvement by adding 1-D convolutional layers between the recurrent layers. Specific results are reported in Section 2.4.

The multi-time step processing converts matrix-vector multiplications into matrix-matrix multiplications. The weight matrix is reused for T time steps with a single parameter fetch from the DRAM, by which the execution time and power consumption can be greatly reduced. The multi-time step computation is shown in Eq. (3), where T is the number of parallelization steps.

$$\begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_T \\ \mathbf{i}_1 & \mathbf{i}_2 & \dots & \mathbf{i}_T \\ \mathbf{f}_1 & \mathbf{f}_2 & \dots & \mathbf{f}_T \\ \mathbf{o}_1 & \mathbf{o}_2 & \dots & \mathbf{o}_T \end{bmatrix} = \begin{pmatrix} \mathbf{W}_z \\ \mathbf{W}_i \\ \mathbf{W}_f \\ \mathbf{W}_o \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \, \mathbf{x}_2 \dots \mathbf{x}_T \end{pmatrix} \tag{2.3}$$

2.3.2 Character-based model

RNN based LMs show quite high performance when compared to statistical n-gram based LMs. When AM employs characters as the output unit, a dictionary or character-level LM (CLM) can be used for reducing the word error rate (WER) [37]. The CLM only supports 30 labels; therefore, the input and output layers are very simple. In addition, the CLM does not have the out-of-vocabulary (OOV) problem. We do not use the word-level LM (WLM) because it has the OOV problem and consumes a large number of parameters at the output softmax layer. Instead of adopting WLM, we use the hierarchical character-level LM (HCLM) for further improving the performance [38]. Note that the HCLM consists of two RNN modules: one operates with the character clock and the other with the word clock. Because the RNN modules operate with the word-clock, the HCLM can show very low bit-per-character (bpc) performance.

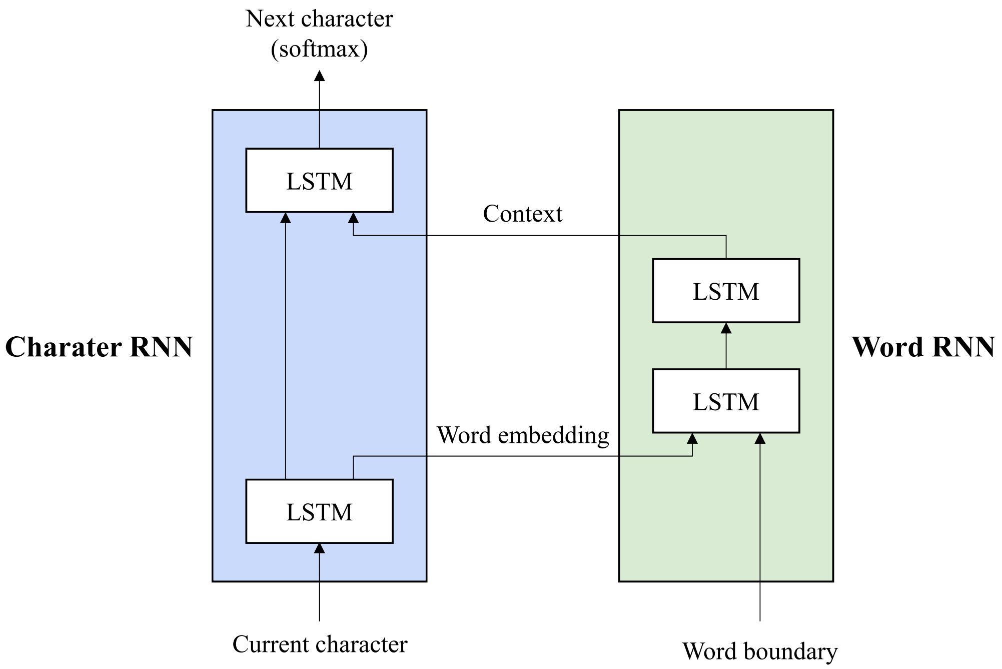


Figure 2.2: The architecture of the hierarchical recurrent neural network model used for language modeling.

The LMs are used in beam search decoding, and the maximum number of LMs that operate simultaneously depends on the beam size, which is between 32 and 128 in this system. This suggests the use of multi-stream parallel processing to improve the execution speed of LMs. Note that multi-stream parallel processing executes multiple sequences concurrently, while multi-time step parallel processing, which is adopted in AM RNN, computes multiple output samples at a time. Thus, conventional RNN, such as LSTM or GRU models, can be used for LM design. We need to save the context of all the LMs during the beam search decoding. GRU has fewer states to keep than LSTM. Therefore GRU can reduce memory consumption in the decoding stage.

2.3.3 Word piece-based model

We have also developed a word piece model based ASR to reduce the complexity further by lowering the frame rate. The word piece model includes very frequently used words, sub-words, and characters [34]. The number of word pieces used in this system is 500 and 1,000. The word piece model does not have the OOV problem because any word can be constructed using sub-words or characters. The performance of the word piece LM can be better than that of CLM because a word piece is usually composed of several characters, which implies an ability to predict longer dependency than CLM. The word piece model is very advantageous for reducing the complexity because the AM with word piece can operate at a slower rate when compared to the AM with character. In addition to the frame rate down-sampling at the convolutional layer, we apply down-sampling in the recurrent layers also. However, word piece model training, especially for AM, demands much more data because there are an increased number of labels.

2.3.4 Decoding

For inference, we find the sequence of label $\mathbf{y}$, which maximizes $Q(\mathbf{y})$ for a given input feature $\mathbf{x}_{1:T}$, by combining the output of AM P_{CTC} and LM P_{LM} as follows

$$Q(\mathbf{y}) = \log(P_{\text{CTC}}(\mathbf{y}|\mathbf{x}_{1:T})) + \alpha \log(P_{\text{LM}}(\mathbf{y})) + \beta|\mathbf{y}| \tag{2.4}$$

The labels can be either characters or word pieces. We use the beam search decoding algorithm for incremental speech recognition as proposed in [37]. The computational complexity of beam search decoding with RNN LM is O(*beam width* $\times$ *transcription length* $\times$ *vocabulary size*). To decrease the search space of decoding, we applied two techniques. First, we skip the decoding process for the current input when the blank output probability is larger than 0.95. This removes unnecessary search caused by blank frames [39]. Second, we sort the AM output probability and conduct decoding for top-k probability labels only. This is especially effective for word piece models because the vocabulary size of word pieces is at least 10 times larger than that of the character level LM. We used $k = 10$ for word piece model.

The most time-consuming part in the decoding is computing the probability of $P_{LM}(c|l)$, which is required in the line 17 and 20 of Algorithm 1. Time complexity of the LM computation is $O(B \times T \times |\Sigma|)$, but the actual computation complexity can be reduced to $O(B \times |l| \times |\Sigma|)$ by reusing the result of LM for same inputs. Skipping consecutive blanks and candidate pruning are applied in line number 5 and 8, respectively. Table 2.1 shows the ratio of skipped repeated blank frames. The number of operations of LM per frame according to the number of candidates are shown in Table 2.2.

2.4 Experimental Results

We present the AM training results on character and word piece models. The decoding is conducted with RNN LMs. In addition, the execution time is analyzed.

2.4.1 Acoustic models

The most critical part of this research is the development of an AM using linear RNN, such as SRU or QRNN. In our early experiments, we failed to obtain good performance by only using SRU or QRNN for recurrent layers. Therefore, we needed to try many different RNN structures, thus each training should not take much time. We constrained the number of parameters to be approximately 12M. The models used for performance evaluation include the conventional LSTM, SRU, Gated ConvNet, and GILR-LSTM [22]. In addition, we trained each model with the 1-D convolution at the input. The width of 1-D convolution is set to 15, which seems to be the optimum number at our experiments. Note that the 1-D convolution considers 7 past (-7) and 7 future (+7) time-steps unless specified otherwise. We used uni-directional models because this implementation is intended for online speech recognition. Bi-directional models for

```
Input: AM output probability matrix $P_{\text{CTC}} \in \mathbb{R}^{|\Sigma| \times T}$, beam width $B$, number
       of search candidates $k$, language model weight $\alpha$, insertion bonus $\gamma$,
       vocabulary $\mathbf{\Sigma}$
1  $\mathbf{A}_{prev} = \{\phi\}$, $P_{\text{CTC}}(blank|\mathbf{x}_{1:0}) = 1$
2  for t = 1 to T do
3      if $P_{CTC}(blank|\mathbf{x}_{1:t-1})$ ¿ 0.95 and $P_{CTC}(blank|\mathbf{x}_{1:t})$ ¿ 0.95 then
4          continue
5      end
6      $\mathbf{A}_{next} = \{\}$, $\mathbf{K}$ = top-$k$ labels in $\mathbf{\Sigma}$ according to value of $P_{\text{CTC}}(\mathbf{c}|\mathbf{x}_{1:t})$
7      for l in $\mathbf{A}_{prev}$ do
8          for c in $\mathbf{K}$ do
9              if c = blank then
10                 $p_{nb}(l) = p_{nb}(l) P_{\text{CTC}}(blank|\mathbf{x}_{1:t})$
11                 $p_b(l) = (p_{nb}(l) + p_b(l)) P_{\text{CTC}}(blank|\mathbf{x}_{1:t})$
12             else
13                 $l^+ = concat(l, c)$
14                 if $c = l_{end}$ then
15                     $p_{nb}(l^+) = p_b(l) P_{\text{CTC}}(c|\mathbf{x}_{1:t}) \gamma P_{LM}(c|l)^\alpha$
16                     $p_{nb}(l) = p_b(l) P_{\text{CTC}}(c|\mathbf{x}_{1:t})$
17                 else
18                     $p_{nb}(l^+) = (p_b(l) + p_{nb}(l)) P_{\text{CTC}}(c|\mathbf{x}_{1:t}) \gamma P_{LM}(c|l)^\alpha$
19                 end
20             end
21             add $l^+$ to $\mathbf{A}_{next}$
22         end
23     end
24     assign top-$B$ of $\mathbf{A}_{next}$ to $\mathbf{A}_{prev}$
25 end
```

Algorithm 1: Prefix beam search in proposed system.

Table 2.1: The ratio of frames whose decoding stages are skipped due to high CTC blank output.

Acoustic model	Downsampling ratio	Percentage of skipping
WSJ - Character	$\times 2$	33.8%
WSJ - Word piece	$\times 4$	44.33%
Libri - Word piece	$\times 8$	20.23%

Table 2.2: The number of LM operations with the varying number of candidates.

Number of candidates	20	30	40	100
LM operations / frame	4.365	4.488	4.593	4.820

i-SRU and LSTM are also included for performance comparison.

We used Wall Street Journal (WSJ) SI-284 training set (81 hours) for the fast evaluation of AMs. A 40-dimensional log mel-frequency filterbank feature was extracted from the raw speech data. The feature vectors were sampled every 10 ms with 25 ms Hamming window. We applied batch normalization [40] to the first two convolutional layers and variational dropout [41] to every output of the recurrent layer for regularization. Adam optimizer [42] was applied for training. We used an initial learning rate of 3e-4, and the learning rate was reduced to half if the validation error was not lowered for consecutive 8 epochs. Gradient clipping with a maximum norm of 4.0 was applied. For comparison, we trained all the models with identical hyper-parameter setting. All the experiments are performed with TensorFlow [43].

The trained models were evaluated on WSJ eval92 set. The beam search decoding was conducted using the same CLM or the same HCLM. The CLM used for decoding consists of two-layer 512-dimensional LSTM. The HCLM has four recurrent layers, where two layers are assigned to the word-level modeling [38]. RNN LM was trained

Table 2.3: WER and CER in percentage on WSJ eval92 test set. Decoding is conducted with RNN CLM and HCLM.

		Greedy		CLM		HCLM	
Model	**Params.**	**CER**	**WER**	**CER**	**WER**	**CER**	**WER**
6x800 SRU	10.62M	26.94	82.56	13.24	29.68	7.94	15.41
6x700 i-SRU	10.92M	12.70	45.22	7.04	18.90	4.90	12.27
6x800 SRU, 1-D conv	10.69M	6.06	22.16	3.48	9.53	**1.97**	**4.90**
6x700 i-SRU, 1-D conv	10.98M	**5.26**	**19.07**	**2.70**	**7.30**	2.01	**4.90**
6x1000 i-SRU, proj, 1-D conv	14.14M	5.85	21.60	3.00	7.80	2.27	5.17
4x600 LSTM	10.85M	7.29	24.88	5.35	14.27	3.70	8.75
4x600 LSTM, 1-D conv	10.88M	6.95	23.57	5.80	15.22	3.10	7.01
4x840 LSTM, proj, 1-D conv	12.01M	7.78	26.80	4.88	12.26	3.36	7.60
6x300 Gated ConvNet	16.38M	8.02	28.65	5.13	13.82	2.98	6.74
4x550 GILR-LSTM	11.34M	8.60	31.99	4.86	13.60	2.66	6.35
4x550 GILR-LSTM, 1-D conv	11.37M	7.15	26.06	4.44	11.92	2.38	5.45
bidirectional models							
6x400 i-SRU, 1-D conv	11.52M	**4.90**	**17.30**	**2.94**	**7.90**	**1.97**	**4.87**
4x350 LSTM	10.70M	5.88	20.17	3.46	9.41	2.57	5.89

Table 2.4: Comparision of the model with non-causal and causal 1-D convolutions. 1-D conv (-a, b) uses a past and b future time-steps to compute the output of the current time step.

	Greedy		CLM		HCLM	
Model	**CER**	**WER**	**CER**	**WER**	**CER**	**WER**
6x700 i-SRU, 1-D conv (-7, 7)	5.26	19.07	2.70	7.30	2.01	4.90
6x700 i-SRU, 1-D conv (-14, 0)	5.70	20.18	3.12	8.47	2.30	5.32
6x700 i-SRU, 1-D conv (-7, 0)	6.10	21.96	2.99	7.69	2.35	5.55

Table 2.5: WER and CER in percentage on WSJ eval92 test set when trained with additional data.

		Greedy		CLM		HCLM	
Model	**Params.**	**CER**	**WER**	**CER**	**WER**	**CER**	**WER**
6x700 i-SRU, 1-D conv	11.0M	4.13	18.02	2.54	**6.04**	1.51	3.73
6x1000 i-SRU, proj, 1-D conv	14.1M	**3.80**	14.70	**2.19**	6.20	**1.48**	**3.70**
4x600 LSTM, 1-D conv	10.9M	4.35	**13.90**	3.72	10.15	2.55	5.92
4x840 LSTM, proj, 1-D conv	12.0M	5.76	20.15	3.54	9.25	2.53	5.79
Deep Speech 2	100M	WER 3.60 with 5-gram LM					

on WSJ LM training text. We randomly selected 5% of WSJ LM training text to the valid set, and another 5% to the test set. The remaining 90% of the text is used for training RNN LM. The RNN LM reported 1.20 of bit-per-character (bpc) on the test set, while the HCLM showed a bpc of 1.07 on the test set. The decoding was conducted with a beam width of 128.

Table 2.3 shows the CER and WER performance of the RNN models trained with the WSJ SI-284 training set. To denote the projection, 'proj' is used for each RNN model. The size of the projection-layer is a half of the RNN dimension. For example, the LSTM with the layer size of 840 employs the projection dimension of 420. The table includes the results of greedy, CLM, and HCLM based decoding. Here, we can find that i-SRU with 1-D convolution performs much better than LSTM. The performance of SRU with 1-D convolution is not much different from that of i-SRU with 1-D convolution. In this table, we can also find that HCLM helps considerably in reducing the CER and WER for all models.

Since the 1-D convolution layers consider the future inputs for improved performance, the output is not generated immediately. We trained i-SRU with causal depth-wise 1-D convolutions and the results are shown in Table 2.4. The performances of some other models that employ different observation windows are also shown. Note that the WER of the model with the causal 1-D convolution is still much lower than that of LSTM.

We compared the train and valid loss curves of some selected models. The peaks in training curves are due to curriculum-like learning scheduling. SRU without 1-D convolution was not trained well. LSTM converged faster than other models but it reached local minimum quickly. Training loss of i-SRU was reduced faster than SRU, while they showed the similar valid loss in the end of training.

Since the amount of data is critical, we further trained two selected models, i-SRU and LSTM, using all the available speaker independent data in the WSJ corpus to improve the WER. This corresponds to approximately 167 hours of speech. The results

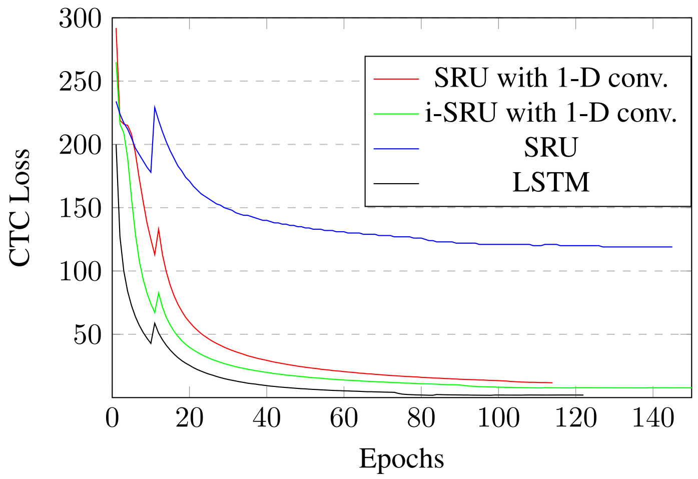


Figure 2.3: Training loss of acoustic models when trained on WSJ SI-284.

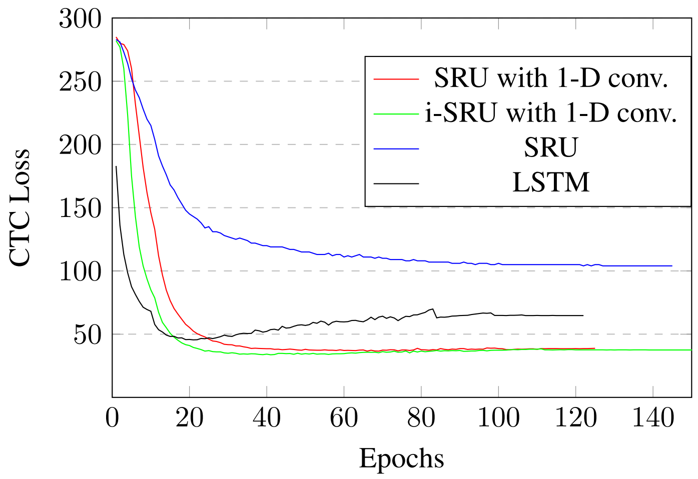


Figure 2.4: Validation loss of acoustic models when trained on WSJ SI-284

Table 2.6: WER and CER on WSJ eval92 when word piece units are used.

	Greedy		WPLM	
Model	**CER**	**WER**	**CER**	**WER**
6x700 i-SRU, 1-D conv	7.37	17.95	6.73	10.50
4x600 LSTM, 1-D conv	9.34	22.56	8.47	15.64
6x700 i-SRU, 1-D conv, additional data	5.47	14.38	3.11	8.28
4x600 LSTM, 1-D conv, additional data	6.57	15.32	4.53	11.48

are shown in Table 2.5. WER of DeepSpeech2 is included for comparison, which is trained with more than 10,000 hours of data and the decoding is performed using 5-gram LM.

2.4.2 Word piece based speech recognition

We trained the i-SRU with 1-D convolutional layer on the word piece model with a vocabulary size of 500. We also trained the LSTM model with the same setting to compare the CER and WER. We added the time-convolution with a stride of two just before the last two recurrent layers. This incurs x4 down-sampling in total. Decoding was conducted under a beam width of 64 with the word piece-level LM. The word piece LM consists of two layers of 512-dimensional LSTM, which has the same structure as that of RNN LM in Section 2.4.1.

The CER and WER for the word piece models are listed in Table 2.6. We noticed that the word piece models show higher WER when compared to the character-based models. One possibility is the data sparsity problem due to the small amount of training data in WSJ SI-284. To relieve this problem, we trained the word piece models using all speaker independent data in WSJ training set, which has 167 hours of data. We could

Table 2.7: Comparison of WER and CER on WSJ eval 92 according to downsampling in the word piece AMs.

	Greedy		WPLM	
Model	**CER**	**WER**	**CER**	**WER**
x2 in conv. layer	7.02	18.95	6.05	10.93
x4 in conv. layer	8.05	20.24	6.55	11.83
x2 in conv. layer, x2 in recurrent layer	7.37	17.95	6.00	10.50
x4 in conv. layer, x2 in recurrent layer	10.30	25.58	7.83	13.99

obtain fairly reduced WER as shown in the table.

The difference between WER and CER in the word piece models is much smaller when compared to that in character based models. This suggests that WER improvement by beam search decoding is less significant in the word piece models. This leads to a smaller beam width in the decoding process, which also reduces the decoding complexity.

One advantage of employing the word piece unit is the possibility of more aggressive down-sampling. We analyzed the effect of down-sampling on WER in Table 2.7. The additional down-sampling is located in the second 2-D convolutional layer or before the last two recurrent layers. The lowest WER is reported when a down-sampling of 2 is applied to both the 2-D convolutional layer and the recurrent layer. Increasing the down-sampling ratio is very beneficial in reducing the decoding complexity.

We also trained our system using a larger dataset, Librispeech Corpus [46]. The AMs were trained with *train-clean-100* and *train-clean-360*. The training hyperparameters were exactly the same with the setting in Section 2.4.1. We trained both the character and word piece models. The word piece vocabulary sizes were 500 and 1,000, respectively. Two-layer 600-dimensional GRU was employed for RNN LM. The results of WERs

Table 2.8: WER and CER on Librispeech *test-clean* . The models are trained on LibriSpeech *train-clean-100* and *train-clean-360*.

		Greedy		RNN LM	
Model	**Params.**	**CER**	**WER**	**CER**	**WER**
4x600 LSTM, character	10.85M	8.49	26.10	7.34	21.80
6x700 i-SRU, 1-D conv, character	10.98M	6.21	20.41	5.66	13.78
6x700 i-SRU, 1-D conv, word piece-500	11.30M	6.72	17.10	4.67	9.98
6x700 i-SRU, 1-D conv, word piece-1000	11.65M	6.62	16.16	4.42	9.61

Table 2.9: WER on Librispeech *test-clean* and *test-other*. The models are trained on all the LibriSpeech train set (960 hours).

Model	**Params.**	**test-clean**	**test-other**	**LM type**
6x700 i-SRU, 1-D conv	12M	9.02	23.60	RNN LM
12x1000 i-SRU, 1-D conv	36M	5.73	15.96	RNN LM
Gated ConvNet [27]	208M	4.8	14.5	4-gram LM
4x1024 bidirectional GRU [44]	75M	5.4	14.7	4-gram LM
Encoder-decoder [45]	150M	3.82	12.76	RNN LM

Table 2.10: Execution time of SRU-AM for 1 second of speech according to the number of parallelization steps.

Parallelization Step	1	2	4	8	16	32
Computation time	1.2129	0.6098	0.3065	0.2064	0.1524	0.1174

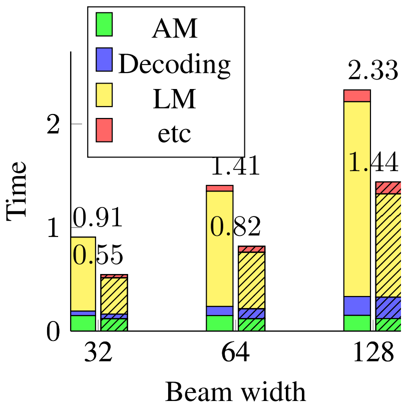
on *test-clean* are shown in Table 2.8. The word piece model shows better performance than the character-level model unlike the WSJ dataset. This shows that the word piece model could be better if larger training dataset is available.

We further trained a large model with all the LibriSpeech training data (960 hours), which includes *train-other-500*. For the large model, we added 1-D convolutions at the input of every two SRU layers. The projection layers are used in each output of SRU layer. The word piece model with a vocabulary size of 1,000 is used. The WERs of the model are reported in Table 2.9. Ours achieves competitive WER compared to other models in recent works. Note that our model is unidirectional and has a far less number of parameters than other ones.

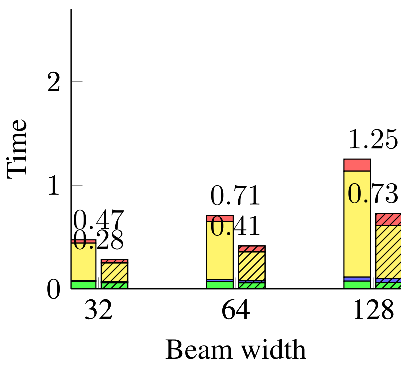
2.4.3 Execution time analysis

We present the implementation results of the proposed speech recognition models on the ARM Cortex-A57 based embedded system. The ARM CPU has 80 KB L1 data cache and 2,048 KB L2 cache. OpenBLAS library [47] is used for the optimization of computation. Table 5.5 shows the execution time of 6x700 i-SRU with 1-D convolution in Table 2.3 according to the number of multi-time steps for parallelization. When the number of parallel steps is 8, the execution time of AM for one second of speech is reduced to 0.2 sec, showing a speed-up of 6 times when compared to the single step execution.

Figure 2.5 shows the execution time estimate of the character and word piece models



(a) Character-level model.



(b) Word piece-level model.

Beam	32	64	128
character	6.24	6.15	6.04
word piece	8.28	8.28	8.26
character (8-bit)	6.47	6.33	6.30
word piece (8-bit)	8.97	8.97	8.96

(c) WER with different beam width.

Figure 2.5: (a, b): Processing time of the speech recognition system for 1 second of speech on the single core ARM CPU. The time is evaluated on the WSJ eval92 dataset. The plot with dashed lines represents the computation time with 8-bit weights. (c): WERs when different beam width is used.

when the beam widths are 32, 64, and 128. The 6x700 i-SRU with 1-D convolution is used for AM of the system. Decoding is conducted with CLM or WPLM explained in Section 2.3.2. We also present the execution time when 8-bit weights are used for computation. For 8-bit implementation, gemmlowp library [48] is employed. The additional down-sampling allows the word piece model to run much faster than the character based model. Furthermore, the word piece model demands a much smaller beam width than the character model. By decreasing the beam width from 128 to 32, the computation time can be reduced to less than half while the difference in WER is only about 0.02%. Therefore, the word piece model is more advantageous for real-time speech recognition.

2.5 Concluding Remarks

Real-time automatic speech recognition (ASR) on embedded CPUs is studied by integrating end-to-end trained acoustic RNN, character or word piece language model RNN, and efficient decoding algorithm. To reduce the DRAM access overhead, we apply multi-frame parallel processing for the AM RNN, and develop high accuracy CTC-trained AM using simple recurrent units (SRUs) combined with 1-dimensional convolution at the input. We develop two ASR models; one employs character-based AM operating at 20 msec frame rate, and the other uses the word piece based AM that operates at the frame rate of 40 msec. The character based model shows very high accuracy on WSJ corpus when combined with the hierarchical character language model. The word piece based model shows x2 of the real-time speed on an ARM CPU mainly due to x4 down-sampling of the word piece AM. This study can be applied to all single stream or small batch-size implementation of ASR regardless of the platform, such as GPU or special-purpose hardware.

Chapter 3

Low-Latency Lightweight Streaming Speech Recognition with 8-bit Quantized Depthwise Gated Convolutional Neural Networks

3.1 Introduction

Deep neural network-based acoustic models have greatly improved the accuracy of automatic speech recognition (ASR) [49, 2, 3]. Neural network-based ASR requires a lot of computations, which demands high memory bandwidth and power consumption for the real-time operation. Server-based implementations are mostly employed to address these issues. However, data transfer between server and edge devices can result in high latency and privacy issues. Also, the connectivity to the network must be guaranteed to use the server-based service. Therefore, it is important to develop an on-device ASR system that can operate in real-time even with limited computing resources [50].

Recurrent neural networks (RNNs), such as an LSTM [9] or GRU [10], have been widely used for acoustic modeling. RNNs have feedback from the output of the previous time-steps and it is advantageous in learning from long sequences such as speech. However, the computation of a single sequence is difficult to parallelize because of the temporal dependency due to the feedback structure. This results in ineffective cache utilization because the weights cannot be reused for multiple time-steps. This especially matters for on-device applications where the cache capacity is often less than the parameter size of neural network models.

Recently, easily parallelizable models such as convolutional neural networks (CNNs) [24] or quasi-RNNs (QRNNs) [20] are actively studied for the sequential tasks. For the speech recognition, fully convolutional networks were trained with the connectionist temporal classification (CTC) and they showed a lower word error rate (WER) than RNN-based models [27, 51]. However, these structures suffer from the large parameter size when a wide structure is adopted for acoustic modeling. Depthwise convolutions can be used to reduce the complexity of a CNN or a QRNN. The models with depthwise convolutions are successfully applied to CTC- [30, 52] and attention-based models [53].

In this chapter, the implementation of an on-device streaming end-to-end speech recognition system with the simple gated convolutional network (SGCN) [52] is proposed. This design is intended for low-cost ARM CPU based implementations supporting SIMD (Single Instruction Multiple Data) instructions. In comparison, SGCN is more advantageous to other architectures when the number of parameters is limited. The model is trained with CTC [35], which is suitable for online decoding. The training method with symmetrical noise injection is applied to obtain high accuracy. 8-bit quantization is employed to further reduce the memory bandwidth and use SIMD instructions. This design is intended for low-cost ARM CPU based implementations supporting SIMD (Single Instruction Multiple Data) instructions. The proposed system features streaming inference, where the intermediate outputs are available before the end of input is given. The 1.2MB end-to-end ASR model is obtained, which operates in 0.2 RTF (five times the speed of real-time) with only a single 900MHz CPU core. The proposed system has a WER lower than 20% on WSJ eval92 without any LM.

This chapter is organized as follows: In Section 2, the structure of SGCN is addressed. In Section 3, the methods to train and quantize SGCN for CTC acoustic modeling are described. The experimental results are shown in Section 4. Section 5 concludes the chapter.

3.2 Simple Gated Convolutional Networks

3.2.1 Model structure

The gated convolutional network (GCN) is a non-recurrent network architecture which has been successfully applied to sequential tasks, such as language modeling [26], translation [24], and speech recognition [27]. GCN employs a gating mechanism similar to the output gate of the LSTM. When the input $\mathbf{x}_t \in \mathbb{R}^D$ is given, the output of GCN $\mathbf{h}_t$ is computed as follows:

$$\mathbf{h}_t = f(\sum_{i=t-W+1+d}^{t+d} \mathbf{V}_i\mathbf{x}_i + \mathbf{b}) \odot \sigma(\sum_{i=t-W+1+d}^{t+d} \mathbf{U}_i\mathbf{x}_i + \mathbf{c}), \tag{3.1}$$

where $\mathbf{V}_i, \mathbf{U}_i \in \mathbb{R}^{D\times D}$ and $\mathbf{b}, \mathbf{c} \in \mathbb{R}^D$ are trainable variables. σ is a sigmoid function for gating. We used ReLU for activation function f. W and d denote the width and delay of GCN. Computation of $\mathbf{h}_t$ does not have any dependencies on $\mathbf{h}_{t-1}$, and the outputs for multiple time-steps can be computed simultaneously.

GCNs can only consider a finite range of context which is limited by the filter width. To apply GCNs to speech recognition, a filter wider than 15 time-steps is often used. The number of parameters and computations for GCN is approximately $2WD^2$ and is proportional to the width. This drastically increases the parameter size of CNN for an on-device application.

Simple gated convolutional networks (SGCNs) employ depthwise convolutions to reduce the parameter size. In SGCN, the width of 1-D convolutions in GCN is reduced to 1. Instead, depthwise convolutions are applied to increase the length of the input context. The depthwise convolution in SGCN consults K neighboring features to consider the correlation between them. The output of SGCN is computed as following equations.

$$x'_{t,k} = \sum_{j=0}^{K-1} \sum_{i=t-W+1+d}^{t+d} w_{i,j} \cdot x_{i,k+j}, \tag{3.2}$$
$$\mathbf{h}_t = f(\mathbf{V}\mathbf{x}'_t + \mathbf{b}) \odot \sigma(\mathbf{U}\mathbf{x}_t{}' + \mathbf{c}).$$

The number of parameters for the convolution and depthwise convolution are $2D^2$ and KWD, respectively. In general, D is a few hundred and K is about 5. Therefore, increasing the width of SGCN has a negligible impact on the total number of parameters.

Figure 3.1 describes the whole structure of SGCN for AM. Two 2-D convolutions are applied to the input features in frequency- and time-axis. The output of the first layer was max-pooled with the factor of 2 for down-sampling. The SGCN layers are stacked on top of the 2-D convolutions. A residual connection is used for every two SGCN layers.

3.2.2 Multi-time-step parallelization

The multi-time-step parallelization merges matrix-vector multiplications for the multiple inputs into a single matrix-matrix multiplication as follows.

$$[\mathbf{x}'_1, \mathbf{x}'_2, ..., \mathbf{x}'_T] = \mathbf{V}[\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T] \tag{3.3}$$

With a single fetch from DRAM, the weights can be reused for the T time-step inputs. The multi-time-step parallelization cannot be applied to the recurrent path in RNN, where the computation of the previous time-step output must be completed before the current time-step computation. If the cache size is smaller than the parameter size, the number of DRAM accesses can be reduced to $1/T$ by applying the multi-time-step parallelization.

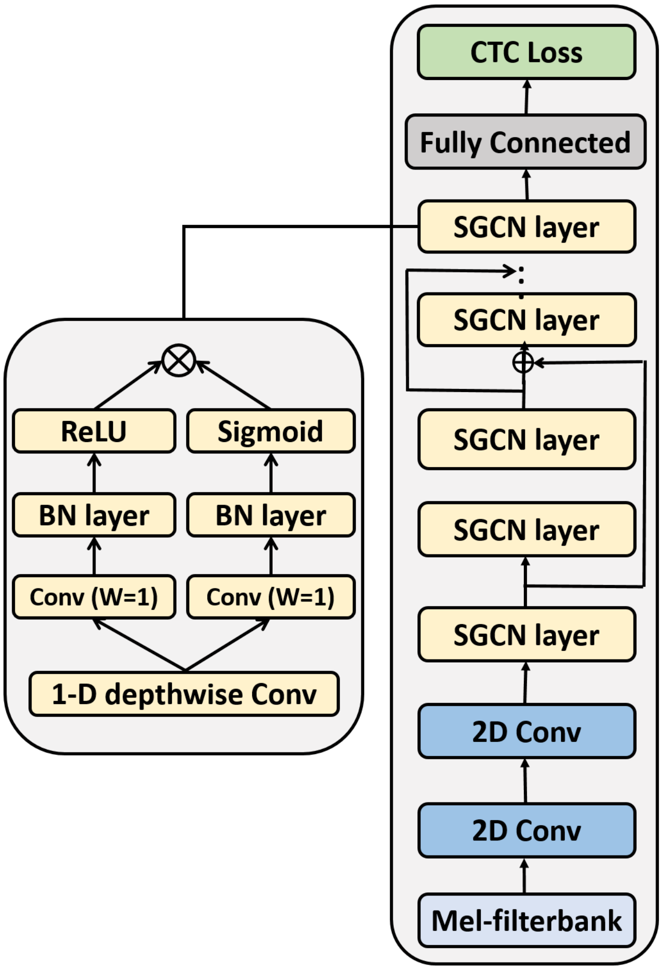


Figure 3.1: A SGCN architecture for acoustic modeling.

Table 3.1: WER (%) of SGCN trained with the symmetrical weight noise injection. SN denotes the symmetrical noise injection. The models are trained on WSJ si-284.

Model	**Params.**	**WER**
12x190 K=5, W=11	1.09M	21.66
12x190 K=5, W=11 + SN	1.09M	19.90
12x300 K=5, W=11	2.24M	18.30
12x300 K=5, W=11 + SN	2.24M	16.87
Jasper 10x3 [51]	201M	13.3

Table 3.2: WER (%) of SGCN trained with symmetrical weight noise injection. Trained on WSJ si-all (147 hours).

Model	**Params.**	**WER**
12x190 K=5, W=11	1.09M	18.18
12x190 K=5, W=11 + SN	1.09M	16.76
12x300 K=5, W=11	2.24M	15.30
12x300 K=5, W=11 + SN	2.24M	12.74

3.3 Training CTC AM with SGCN

3.3.1 Regularization with symmetrical weight noise injection

To obtain higher recognition accuracy, the symmetrical noise injection is applied during training. Training with noise injection is used to find wider local minima in the loss surface and avoid overfitting [54, 55]. In the preliminary experiments, applying symmetrical noise injection for training helps faster convergence than using only a single noise. Conventional stochastic gradient descent updates the parameters of the model $\mathbf{w}$ as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{B}\sum_{i=1}^{B}\nabla L_i(\mathbf{w}_t), \tag{3.4}$$

where B is the number of batches in the training set. For the symmetrical weight noise injection, $\widetilde{\mathbf{w}}_{t+} = \mathbf{w}_t + \alpha\mathbf{n}_t$, and $\widetilde{\mathbf{w}}_{t-} = \mathbf{w}_t - \alpha\mathbf{n}_t$ are used for training. Weight noise $\mathbf{n}_t$ is an uniformly distributed random vector. The magnitude of the noise is determined from the standard deviation of the weight. We used 0.05 for the scale factor α.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{2B}\sum_{i=1}^{B}\nabla(L_i(\widetilde{\mathbf{w}}_{t+}) + L_i(\widetilde{\mathbf{w}}_{t-})), \tag{3.5}$$

3.3.2 8-bit quantization

To utilize 8-bit SIMD instructions and lower the memory bandwidth, the weights and activations of the SGCN model were quantized. TensorFlow Lite [56] was used for quantization. The quantization step sizes are determined by the minimum and maximum values. Real values are mapped to quantized values as follows:

$$Q(\mathbf{x}) = (\mathbf{x} - offset)/scale + 128) \tag{3.6}$$

$$offset = (min(\mathbf{x}) + max(\mathbf{x}))/2 \tag{3.7}$$

$$scale = (max(\mathbf{x}) - min(\mathbf{x}))/255 \tag{3.8}$$

Table 3.3: WER (%) evaluated on the WSJ eval92 test set.

Model	**Params.**	**WER**
12x190 K=5, W=11, 1200ms latency	1.09M	16.76
12x190 K=5, W=11, 200ms latency	1.09M	18.46
12x190 K=3, W=11, 200ms latency	1.04M	19.28
12x190 K=5, W=9, 200ms latency	1.07M	19.60
12x190 K=3, W=11, 200ms latency	1.02M	19.70
12x180 K=5, W=11, 200ms latency	1.00M	19.77
12x180 K=3, W=11, 200ms latency	0.94M	20.16
4x160 uni. LSTM [52]	0.96M	31.2
4x160 uni. LSTM + d.w. conv. [52]	0.97M	24.1
12x300 K=5, W=11, 1200ms latency	2.24M	12.74

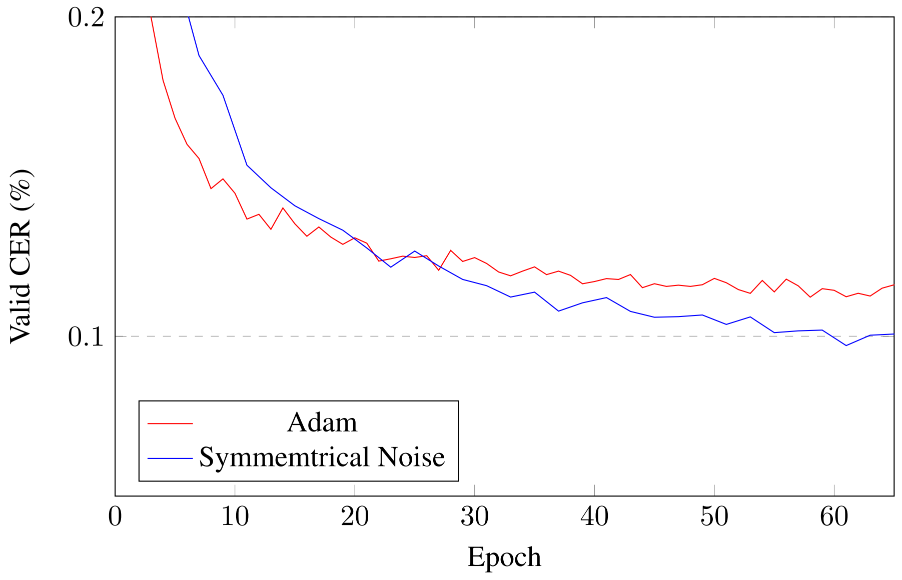


Figure 3.2: Validation CER curve of the SGCN trained on WSJ si-284.

Retraining-based quantization was applied to reduce the accuracy drop of the quantized model. First, SGCN models were trained in the floating-point domain. After the floating-point model was converged, the model was retrained with quantized weights and activations. The minimum and maximum values for quantization were obtained during retraining. For retraining, the learning rate schedule identical to that used for training the floating-point model was utilized. It was found that weight decay was critical for reducing the performance gap between floating-point and quantized models when residual connection and depthwise convolution were employed together.

3.4 Experimental Results

3.4.1 Experimental setting

SGCN was trained with connectionist temporal classification (CTC) algorithm. The Wall Street Journal (WSJ) Corpus [57] was used to train the models. We used two subsets of WSJ Corpus for training, one is si-284, which is the standard subset for training,

Table 3.4: Computation time measured on 900MHz ARM CPU. Only a single core is used for evaluation. RTF denotes the real-time factor.

Model	**Weight**	**Activation**	**Model Size**	**RTF**
12x190 SGCN K=5, W=11	32-bit float	32-bit float	4.30MB	0.469
12x190 SGCN K=5, W=11	8-bit int	32-bit float	1.16MB	0.520
12x190 SGCN K=5, W=11	8-bit int	8-bit int	1.16MB	0.194
12x190 SGCN K=3, W=11	8-bit int	8-bit int	1.08MB	0.191
12x190 SGCN K=5, W=9	8-bit int	8-bit int	1.12MB	0.183
12x180 SGCN K=5, W=11	8-bit int	8-bit int	1.03MB	0.174

Table 3.5: WER (%) of the SGCN before and after quantization.

Model	**WER**
12x190 K=5, W=11, 200ms latency	18.46
12x190 K=5, W=11, 200ms latency, quantized	19.75

and the other is si-all which includes all the speaker-independent data in the corpus. The amount of data in si-284 and si-all are 81 hours and 147 hours, respectively. The utterances with verbalized punctuations were excluded from training. A 40-dimensional log Mel frequency filter-bank with delta and delta-delta is used for the input features.

All the SGCN models in this chapter were trained using the following hyperparameters. For training, a learning rate of 3e-3 was chosen. If the validation error did not decrease for 8 epochs, the learning rate was decayed by 0.2. After the learning rate decayed six times, the training was concluded. The Adam optimizer [42] was employed for training. A model with the lowest validation WER over the whole training processes was selected.

3.4.2 Results on WSJ eval92

The results of training with and without the symmetrical noise injection are shown in Table 3.1 and 3.2. The word error rate (WER) was evaluated on WSJ eval92. 12x190 model denotes that 12 SGCN layers with $D = 190$ are stacked. The symmetrical noise injection consistently improved the performance regardless of the size of the model or training data. For comparison, the WER of recent convolutional CTC model, Jasper [51], was also shown. Considering the number of parameters, the SGCN model shows competitive accuracy.

Table 3.3 shows the WER of SGCN models with various architectures. The models are trained on WSJ si-all. The symmetrical noise injection was applied for training all the models. For the 200ms latency model, we used $d = 5$ for the last 2 SGCN layers and $d = 0$ for the others. All the SGCN layers in the 1,200ms used $d = 5$. When K, W, or dimension of layers were decreased from the 12x190 K=5, W=11 SGCN model, WER dropped accordingly. In our experiments, dimension of the layers has the largest impact on recognition performance. For comparison, the WER of LSTMs with similar model size are shown. The increase of the model size to 2.24M results in the decrease of WER from 16.76 % to 12.74%. This suggests that the SGCN model can scale-up to a large size model.

3.4.3 Implementation on the embedded system

For an on-device implementation, the 12x190 K=5, W=11, 200ms latency model is used. The WER of the model after quantization is reported in Table 3.5.

Table 3.4 reports the computation time and memory size of the SGCN AM. The computation time is measured on Raspberry Pi 2 model B. The system has 900MHz ARM Cortex-A7 CPU and 1GB DRAM. The CPU has 256KB L2 cache. For the results in Table 3.4, 20 frames, which corresponds to 200 ms, of the inputs are computed at

Table 3.6: RTF with the different number of parallelization steps.

Time steps	200 ms	400 ms	3000 ms
RTF	0.194	0.188	0.179

Table 3.7: Percentage of computation time per the type of operations.

Operation Type	**Percentage**	**Cumulative Sum**
Conv.	70.89%	70.89%
Depthwise Conv.	21.06%	91.95%
Dense	0.46%	92.41%
Etc. (Activation, ...)	7.59%	100%

a time. RTF can be reduced if the number of time steps for parallelization increases because of memory access reduction. RTF with different parallelization time-steps is shown in Table 3.6. There is a trade-off between RTF and latency to collect inputs for parallelization.

Table 3.7 shows the computation time for each block. TensorFlow Lite profiler was used to measure the computation time. As analyzed in Section 3.2.1, the most time-consuming operation is the convolutional operation. 71 % of the computation time was dedicated to the convolutions, while the depthwise convolutions occupied 21%.

3.5 Concluding Remarks

In this chapter, we proposed the on-device streaming ASR system which operates in 0.2 RTF with the 900MHz CPU and 1.2MB memory footprint. The WER on WSJ si-284 is 19.84%, and can be lowered if rescoring with LM is applied. The system is promising for applications where the computation and power budgets are very limited, such as a keyword spotting or always-on speech recognizer.

Chapter 4

Effect of Adding Positional Information on Convolutional Neural Networks for End-to-End Speech Recognition

4.1 Introduction

Many automatic speech recognition (ASR) algorithms employ recurrent neural networks (RNNs) because of their ability to recognize sequences [2, 3, 49]. In particular, attention-based models are prevalent for ASR [4], and they usually employ RNNs for the encoder and decoder. However, RNN-based models are very difficult to parallelize, which results in severe restrictions when implementing them on graphics processing units (GPUs) or embedded devices for low power and high speed. The efficiency of implementation is very low, especially when the batch size is very small.

Recently, non-recurrent structures, such as convolution [24] and self-attention [28], have actively been studied for application to sequential tasks for the sake of computational efficiency. Because non-recurrent structures can process multiple input frames at a time, the number of parameter accesses from DRAM can be greatly reduced. In particular, for speech recognition, convolutional neural networks (CNNs) [27, 51] and self-attention networks [58] have been successfully applied to attention-based models and have shown lower word error rate (WER) than RNN-based models. We focus on convolution-based models that require only a limited receptive field size for the input. Note that using a limited-length input for speech recognition is very advantageous for low-latency system design [59]. Although we focused on attention models that are not capable of online inference, the proposed method can be extended to streaming inference when local attentions [29, 60] are applied [61].

Convolutional models for speech recognition often require a large receptive field length to observe a long input context. Note that the receptive field size is determined by the filter length in each layer and the depth of the model. WER increases drastically when the receptive field length is not sufficient. This is mainly due to looping or skipping problems, which are frequently observed when the encoder of the attention model contains similar outputs at different time-steps [62]. Employing a large filter size can help solve this problem, but it demands a large parameter size or computational overhead. Depth-wise convolutions can be used to reduce the parameter size overhead, but they cannot solve the large intermediate memory requirement and delay problem for the input [53, 63, 52]. We consider that the high error rate of models with small receptive fields is caused by the time-invariant property of convolution. When similar pronunciation is repeated in the input speech, a convolutional encoder also yields very close output values. This property can be helpful in terms of generalization, but it results in unstable attention because the model cannot distinguish similar values at different time steps.

In this study, we analyzed the error pattern when convolutional models only have small receptive field sizes. Then, we showed that the recognition accuracy of small receptive-field models can be improved by adding the simplest form of positional encoding, which is used in the Transformer architecture [64]. The encoder output is visualized to prove the effectiveness of positional information. The proposed method improves the accuracy of attention models with convolutional encoders, especially when the models have small filter sizes. We achieved 10.60% WER on TED-LIUMv2 using the single end-to-end model.

This chapter is organized as follows: In Section 2, we review the related works on using positional information for attention-based models. In Section 3, the model structure and experimental setup are described. Experimental results are shown and discussed in Section 4. Section 5 concludes the chapter.

4.2 Related Works

The effect of positional information on CNNs for image recognition was recently studied in [65]. That study showed that a CNN with a large receptive field size inherently learns positional information; the results of recognition rely heavily on positional information. They showed that a sufficient receptive field size and zero-padding are required for convolutional models to learn positional information.

External positional information has been applied since early convolutional models for sequential tasks. Trainable positional embedding vectors are added to word embeddings for training convolutional language models [24]. This approach is not suitable for speech recognition, where the length of the inputs is longer and varies much by data. Sinusoidal positional encoding has recently been proposed with Transformer architecture [28]. Sinusoidal positional encoding can be effective even when the input sequence is longer than the training data.

Positional information is often considered to help stabilize training attention weights. Attention feedback [66] and location-based attention [64] use which part of the encoder output was attended in the past time step to compute current attention weights. Soft-window pretraining uses auxiliary loss that encourages attention weights to be aligned with the input time steps [53]. Through our experimental results, we show that applying positional encoding has a similar effect to these methods without modifying the model structure or training processes.

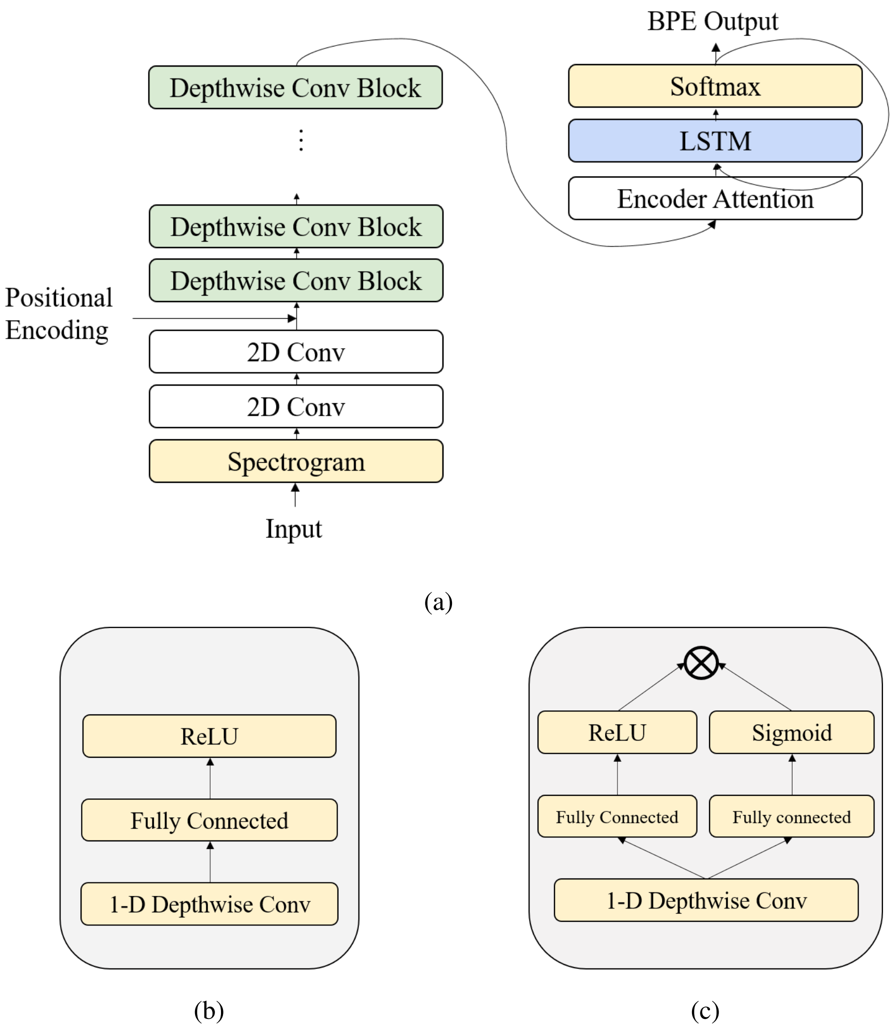


Figure 4.1: (a) The attention-based model with 1-D depthwise convolutions and positional information for encoder. (b) A depthwise convolutional block and (c) a block with gating structure.

4.3 Model Description

The attention-based speech recognition model is based on [45], except that 1-D depth-wise convolutional layers are used for the encoder. Each layer is computed as follows:

$$\begin{aligned} x'_{t,k} &= \sum_{i=-(T-1)/2}^{(T-1)/2} W_{i,k} \cdot x_{t+i,k} \\ \mathbf{h}_t &= f(\mathbf{V}\mathbf{x}'_t + \mathbf{b}), \end{aligned} \tag{4.1}$$

where f is the activation function, and $\mathbf{W} \in \mathbb{R}^{T \times D}, \mathbf{V} \in \mathbb{R}^{D \times D}$ are trainable variables. T and D denote the width and output dimensions of a convolution, respectively. For the encoder, we applied two 2-D convolutions to the input features in the frequency- and time-axis. The 1-D depth-wise convolutional layers are stacked on top of the 2-D convolutions as shown in Figure 4.1. We used a residual connection for every two convolutional layers. Layer normalization [67] was applied after residual connections for better convergence.

In the decoder, the attention weights $\alpha_{i,t}$ and energies $e_{i,t}$ for the encoder time-step t and decoder step i are computed as:

$$\begin{aligned} e_{i,t} &= \mathbf{v}_e^T \cdot \tanh(\mathbf{W}[\mathbf{s}_i, \mathbf{h}_t, \beta_{i,t}]) \\ \boldsymbol{\alpha}_i &= \text{softmax}(\mathbf{e}_i), \end{aligned} \tag{4.2}$$

where v is a trainable vector, $\mathbf{W}$ is a trainable matrix, $\mathbf{s}_i$ the current decoder state, and $\mathbf{h}_t$ denotes the output of the last layer of the encoder. $\beta_{i,t}$ is the attention weight feedback which is defined as:

$$\beta_{i,t} = \sigma(\mathbf{v}_\beta^T \mathbf{h}_t) \sum_{k=1}^{i-1} \alpha_{k,t} \tag{4.3}$$

The attention context vector is given as:

$$\mathbf{c}_i = \sum_t \alpha_{i,t} \mathbf{h}_t. \tag{4.4}$$

The decoder is a single-layer long short-term memory [9] (LSTM) that is computed as follows:

$$\mathbf{s}_i = \text{LSTM}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_{i-1}). \tag{4.5}$$

Usually, the positional encoding vector in Transformer architecture [28] is added to the input of the encoder. For Transformer architecture in speech recognition, a linear transformation is often applied to positional encoding before being added to the input [58]. In our experiments, the best performance was obtained when the positional encoding vector was concatenated to the output of 2D convolutions. We also tried adding or concatenating it to the output of the convolutional encoders, but it made training diverge in the initial stage. We used the positional encoding vector as proposed in [28], which is computed as follows:

$$\begin{aligned} pe_{i,2k} &= \sin(i/10000^{(2k)/Dmodel}) \\ pe_{i,2k+1} &= \cos(i/10000^{(2k+1)/Dmodel}) \end{aligned} \tag{4.6}$$

4.4 Experimental Results

The experiments were performed using the RETURNN framework [68]. TED-LIUM release 2 [69] was used for training, which contains 200 hours of speech. We followed the data preprocessing pipeline used in [70]. We used a 40-dimensional mel-frequency cepstral coefficient for the input features, which were extracted every 10 ms with a 25 ms window size. Byte-pair encoding (BPE) [71] with a vocabulary size of 1K was used for the output labels. The layer-wise pretraining was only applied to LSTM-based models because it lowers the accuracy when used for convolutional models. The decoder was a single-layer 1000-dimensional LSTM algorithm for all models. The configuration files for the experiments are available online.[1]

4.4.1 Effect of receptive field size

The experimental results of the convolutional models on TED-LIUMv2 are shown in Table 5.2. A 15x2048 model denotes that 15 SGCN layers with $D = 2048$ are stacked. We trained the 15x2048 convolutional model while changing the filter width

[1]`https://github.com/car3936/returnn-exp-jinh`

Transcription (WadeDavis_2003_305.16_327.12)

... depend they have a **curious** language and marriage rule which is called linguistic exogamy you must marry someone who speaks a different language and this is all rooted in the mythological past yet the **curious** thing is in these long houses where there are six or seven languages spoken

Without positional encoding

... depend they have a **curious** things and these long houses where they're six or seven languages spoken

With positional encoding

... depend they have a **curious** language and marriage rule which is called linguistic exotic me you must marry someone who speaks a different language and this is all rooted in mythological past get the **curious** things and these long houses were there six or seven languages spoken

Figure 4.2: The original transcript and the decoded results with and without the positional encoding.

Table 4.1: TED-LIUM release 2 results of the models with different filter size. Pos. denotes that positional encoding is applied.

Model	**WER [%]**	
	dev	test
Conv. 15x2048 (T=3)	23.01	18.41
Conv. 15x2048 (T=5)	18.06	15.18
Conv. 15x2048 (T=7)	17.03	14.75
Conv. 15x2048 (T=11)	15.18	12.95
Conv. 15x2048 (T=15)	15.41	13.19
Conv. 15x2048 (T=3) + Pos.	15.75	13.37
Conv. 15x2048 (T=5) + Pos.	15.24	12.58
Conv. 15x2048 (T=7) + Pos.	15.57	13.15
Conv. 15x2048 (T=11) + Pos.	14.78	12.58
Conv. 15x2048 (T=15) + Pos.	14.87	13.14

of depthwise convolutions. We applied max-pooling with a size of 2 for the initial three convolutional layers. All the models have approximately 84M parameters regardless of the filter width because the number of parameters for depth-wise convolution is very small. The results show that positional encoding improved the accuracy of models consistently, especially for those with small filter sizes. Fig. 5.2 shows the WER of the models with different receptive field sizes. The length of the receptive field was calculated as $(W{-}1) \times 80ms \times \#layers$, where W is the filter width. The WER of the models without positional encoding sharply increased when the receptive field length was reduced. In comparison, when positional encoding was employed, the WER was not significantly affected by the length of the receptive field. This result clearly demonstrates the effect of positional encoding on CNN-based models.

We compared the decoding results of two CNN-based models with and without

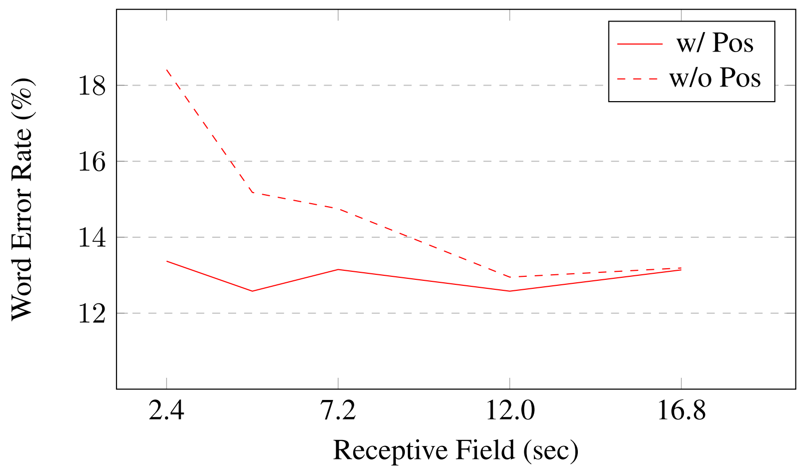


Figure 4.3: Test WER on TED-LIUMv2 comparing models with different receptive field size.

positional encoding. As shown in Fig. 5.3, the original transcription possesses two occurrences of ‘curious’ approximately 30 words apart. The decoding result without positional encoding yields a shortened sentence that skips the words between the two occurrences of ‘curious’. However, decoding with positional encoding faithfully shows all the words. Such skipping occurred frequently over the entire test set. Fig. 4.4 shows a histogram of the number of errors according to the lengths of the transcriptions. We plot the results of the LSTM and the convolutional models with and without positional encoding. The number of errors differed significantly in longer sequences that are more likely to contain repeated words. This suggests that the convolutional model is vulnerable to skipping problems, and applying positional encoding can alleviate this issue.

4.4.2 Visualization

We used principal component analysis (PCA) [72] to analyze the effect of positional encoding on the output of the encoder. Fig. 5.5 shows a visualization of the encoder outputs of the convolutional models. Each point corresponds to a single time-step output

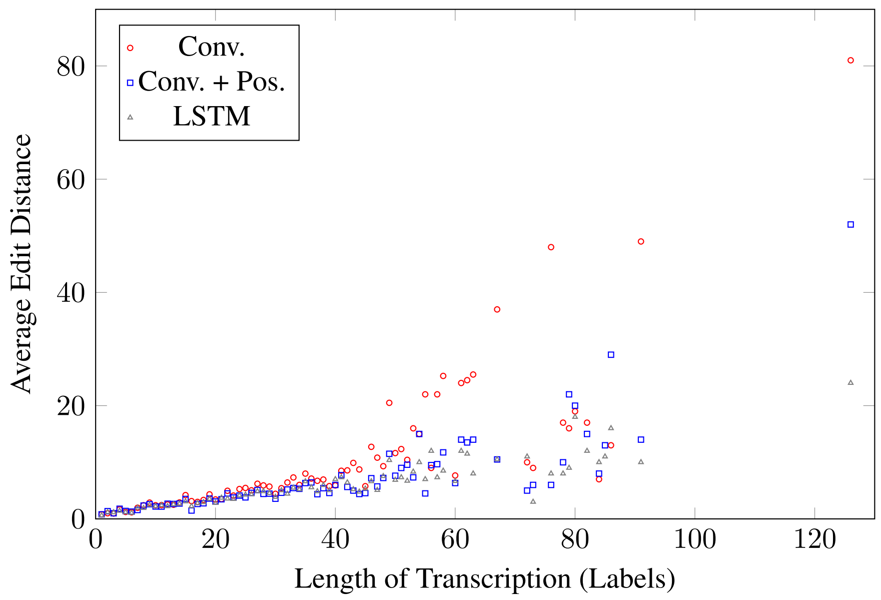


Figure 4.4: The average edit distance of test set according to the length of transcription.

of the encoder. We focused on the results of the 92-94th and 214-216th step outputs, which correspond to the first and second occurrences of ‘curious’, respectively. In Fig. 5.5 (a) and (c), the encoder output of the convolutional models has a similar component when the input speech contains repeated pronunciation. When positional information was applied, the encoder outputs were located at a distance, as shown in Fig. 5.5 (b) and (d). This clearly shows that positional encoding makes the outputs discriminative when similar words are applied.

The attention energy $\mathbf{e}_i$ and the attention weight $\boldsymbol{\alpha}_i$ in Eq. (3) is plotted in Fig. 5.4. The horizontal and vertical axes correspond to the decoder and encoder time steps, respectively. The label from transcription is given to the decoder every step for plotting the attention energy, while the previous output is used for the attention weight. In Fig. 5.4 (a), the energy has a high value in the area indicated by the red box. This causes a misalignment of attention in the inference time and results in skipping, as shown in Fig. 5.4 (b). In Fig. 5.4 (c), the energy is more concentrated around the diagonal components compared to Fig. 5.4 (a). This is a desired property when training the attention-based

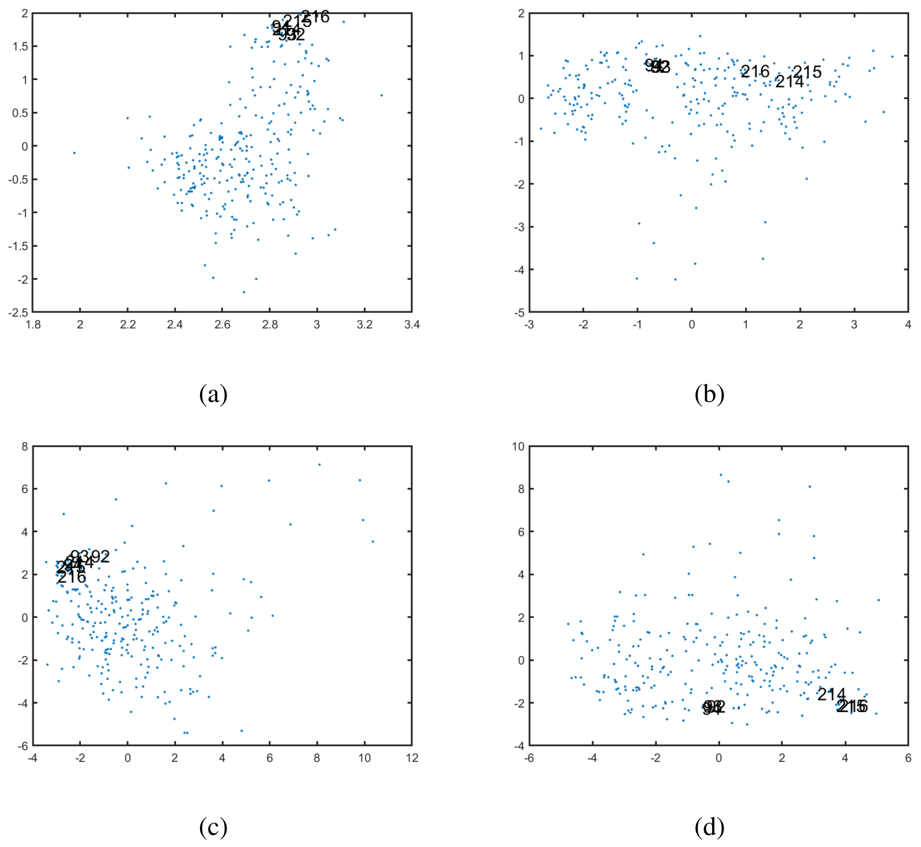
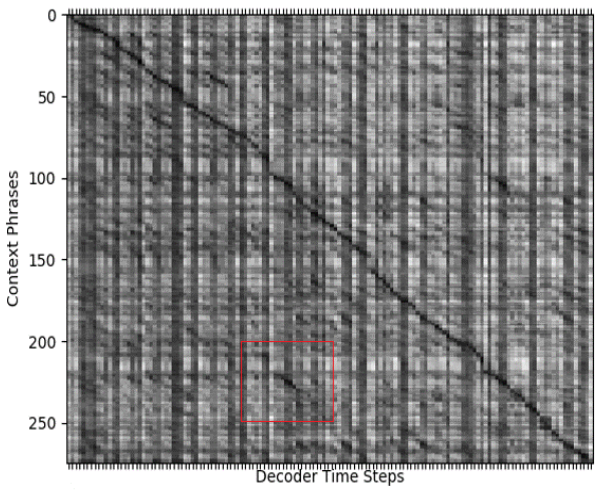
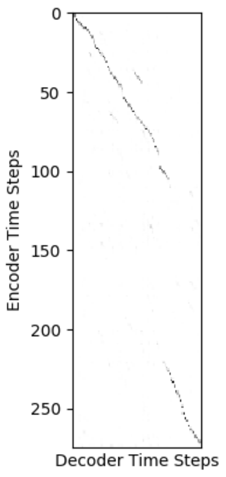


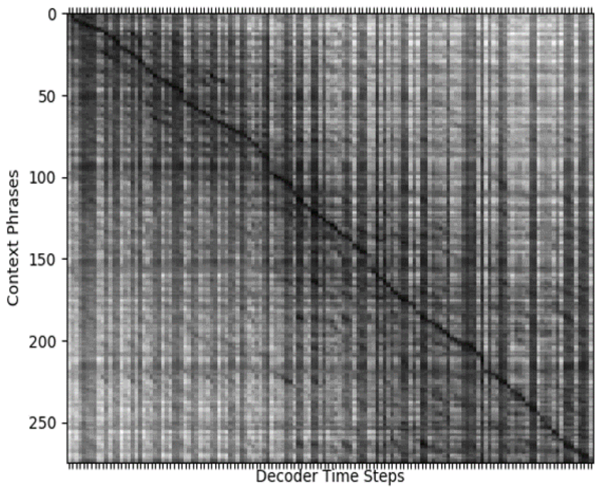
Figure 4.5: The visualization of encoder output using PCA. The first two principal components are used for visualization. The points of 92-94th and 214-216th steps are indicated with text, which correspond to pronunciation of the word ‘curious’. (a) filter width = 3 (b) filter width = 3 with positional encoding. (c) filter width = 5 (d) filter width = 5 with positional encoding.



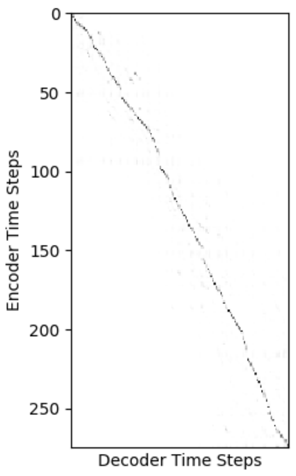
(a)



(b)



(c)



(d)

Figure 4.6: Attention energy $\mathbf{e}_i$ **(left)** and weight $\boldsymbol{\alpha}_i$ **(right)** of models (a) (b) without and (c) (d) with positional encoding are shown. Darker pixels indicate higher values.

model, which prevents the model from diverging in the initial stage of training [53].

4.4.3 Comparison with other models

Table 5.4 shows the results of the convolutional models with large parameter sizes. The results of LSTM- and self-attention-based models are also shown for comparison. The experimental results show that using positional encoding improves the accuracy of deeper structures. Note that bidirectional LSTM and self-attention models consider the entire context of the input, which is not much desired for deployment. Unidirectional LSTM has higher WERs than convolutional models with a comparable number of parameters.

The proposed method can be applied to other convolutional structures. We tried gated convolution [27], which has been successfully applied to speech recognition tasks. With gated convolution, we achieved a 10.60% WER on the TED-LIUM v2 test set.

4.5 Concluding Remarks

In this study, we demonstrated that convolutional models with small filter sizes lack the ability to identify positional information, which incurs looping or skipping problem in end-to-end speech recognition. By adding explicit positional encoding, we prevented severe performance degradation of models with small receptive fields. The proposed method does not require any modification to model structures or training algorithms. It also has almost no computational overhead. Since convolutional encoders support fast training and inference, the proposed method is suitable for developing an on-device low-power speech recognition system.

Table 4.2: Experimental results of convolutional models with different sizes. LSTM and Transformer models are shown for comparison. Decoder is a single-layer 1000-dimensional LSTM for all the models.

Model	**WER [%]**		**Params.**
	dev	test	
Bidir. LSTM 6x1024 [70]	11.7	10.5	161M
Transformer [70]	14.7	12.5	100M
Bidir. LSTM 6x1024	12.65	10.57	161M
Bidir. LSTM 6x1024 + Pos.	17.70	12.35	161M
Unidir. LSTM 6x1536	16.78	14.42	127M
filter width = 3			
Conv. 15x2048	23.01	18.41	84M
Conv. 15x2048 + Pos.	15.75	13.37	84M
Conv. 25x2048 + Pos.	14.51	11.89	127M
Gated Conv. 35x1024 + Pos.	14.94	12.73	80M
Gated Conv. 35x1536 + Pos.	13.02	11.04	229M
filter width = 5			
Gated Conv. 35x2048	14.14	11.16	313M
Gated Conv. 35x2048 + Pos.	12.81	10.60	313M

Chapter 5

Convolution-based Attention Model with Positional Encoding for Streaming Speech Recognition

5.1 Introduction

Recently, the performance of automatic speech recognition (ASR) systems has been significantly improved by end-to-end neural network techniques [3]. Theses end-to-end models include connectionist temporal classification (CTC)-based models [1, 2], attention-based models [3, 4], and recurrent neural network (RNN) or Transformer transducers [5, 6]. Usually, neural network-based speech recognition models have a large number of parameters. These models typically require a huge number of arithmetic operations for *inference*, which requires excessive memory access. Power consumption during the inference is also a major challenge. As a result, cloud-based approaches powered by high-performance servers have been usually adopted for commercial speech recognition systems. Nevertheless, the shortcomings of these cloud-based approaches include the network latency, the server operating costs, privacy concerns due to the transfer of users' private data to the server, and so on. Therefore, on-device speech recognition systems have drawn a lot of attention to overcome the aforementioned problems of the could-based systems. In this chapter, we focus on the attention-based model which shows good speech recognition accuracy without external language models (LMs).

The main bottleneck of on-device neural network-based ASR is the memory access overhead [73]. A typical neural network-based ASR model requires around 1 to 10 Giga arithmetic operations per second. The computational cost in this range is usually not a critical issue these days even for single-core CPUs when optimized with single instruction multiple data (SIMD) operations. Nonetheless, the cache misses are still major challenges. Usually the speech recognition model is larger than 100 MB, which is more than the typical cache size of embedded CPUs. For example, ARM Cortex-A57 has a 2 MB L2 cache at most. When the RNN-based models are employed, which may be the most widely used structure for speech recognition, the parameters should be fetched from the DRAM at every time step due to its feedback structure. Each time step, therefore, these RNN-based models incur continuous cache misses. This problem may be circumvented by batch or multi-stream parallel processing in server-based implementations. In on-device implementations, however, only single stream is available for computation since it usually targets a single user.

To remedy the memory access bottleneck in on-device systems, parallelizable structures including convolution [24] and simplified recurrent structures [20, 74, 22] have been actively studied. Self-attention [28] also has advantages in parallelization. Multiple time-step parallel processing can be applied for these architectures where multiple consecutive frames are computed concurrently with a single weight fetch from DRAM. The number of DRAM accesses can be reduced in proportion to the number of parallelization steps. However, these structures often require a large receptive field size for speech recognition, which increases the number of parameters and computations. Also, the large receptive field increases the working memory size for streaming inference since intermediate results of previous time-steps should be kept.

Table 5.1 compares the working memory sizes of LSTM, convolution, and self-attention-based models. For the convolution, the last $k-1$ frames of the output must

Table 5.1: Memory size of intermediate buffer required for online speech recognition. L is the sequence length, D is the dimension of layer, k is the filter width of convolution. For speech recognition, L is usually a few hundred, while k is about ten.

Model	Working Memory for Streaming	Typical Value	Intra-Sequence Parallelism
LSTM	$2N_{LSTM}D$	98KB	X
Convolution	$kN_{Conv}D$	245KB	O
Self-attention	$LN_{SA}D$	12MB	O

be stored when the filter size is k, while RNN only needs to store the current cell and output state vectors in each layer. Note that the filter size k is usually around 10 $\sim$ 15, but we intend to reduce it to 3. Self-attention requires keeping all the intermediate values from the beginning, which makes its streaming inference difficult on memory-constrained devices. Although time-restricted attention [75] can reduce the memory usage, standard self-attention is often adopted for attention-based speech recognition [58, 76]. Depthwise convolutions are often applied to reduce the computational overhead [63, 52]. Nevertheless, this approach cannot reduce the working memory size. Thus, in this work, we adopt convolutions with limited filter size for on-device applications with limited memory.

The accuracy of convolutional models decreases drastically when the receptive field length is reduced. This degradation mainly comes from the *looping* or *skipping* problems in the output label, which are frequently observed when the encoder of the attention model contains similar outputs at different time-steps [62]. We believe that this phenomenon is caused by the time-invariant property of convolution. When similar pronunciation is repeated in the input speech, a convolutional encoder also yields very close output values. This property can be helpful in terms of generalization, but it results in unstable attention because the model cannot distinguish similar values given at different time steps.

In this chapter, we develop an on-device ASR with convolution-based neural networks for the encoder. Especially, it is shown that the recognition accuracy of small receptive-field models can be improved by adding the simplest form of positional encoding, which is used in the Transformer architecture [64]. The comparison of decoding results and attention weights are shown when the positional information is used or not. The encoder output is visualized with principal component analysis to prove the effectiveness of positional information. The proposed method improves the accuracy of attention models with convolutional encoders, especially when the filter size of the models is small. Along with the monotonic chunkwise attention [29], the proposed method is applied to develop online speech recognition models.

This chapter is organized as follows: In Section 2, we review the related works on end-to-end neural network models for online speech recognition. In Section 3, the model structure and experimental setup are described. We present experimental results including analysis on execution time and speech recognition accuracy on TED-LIUMv2 in Section 4. Section 5 concludes the chapter.

5.2 Related Works

Recently, there have been a lot of researches to develop on-device speech recognition systems with real-time streaming recognition capabilities on resource-limited hardware. CTC [35] and Transducer-based models [50, 77] are often used for online speech recognition due to the causality in the decoding. Attention-based models often show better speech accuracy than the other models. However, conventional global attention mechanism cannot be employed for online speech recognition since it requires the encoder output obtained from the entire input sequence to start decoding. The local attention approaches [29] have been proposed for streaming inference. These approaches have been successfully employed for on-device online speech recognition [61, 78].

Recurrent neural networks such as LSTM [9] or GRU [10] have been often employed for speech recognition tasks due to their advantages in sequence learning. However, the computation of the feedback structure of RNNs has inefficient cache utilization since the weights cannot be reused for multiple-time steps. Therefore, non-recurrent structures such as Quasi-RNN [20], convolutions [27] or self-attention [28] are often employed for on-device streaming speech recognition applications. The RNNs often consider the unlimited future context or recognition accuracy decreases when the unidirectional structure is applied [79]. The self-attention mechanism is also actively employed for end-to-end speech recognition [58, 80, 81, 82, 83]. The masked self-attention is applied to streaming end-to-end speech recognition [76].

Convolutional architectures are often preferred for low-latency speech recognition system [59, 31]. Convolutional neural networks are applied to various architectures for ASR tasks [27]. Attention models with convolutional encoder for speech recognition is studied in [53]. The convolutional model for speech recognition often uses a large receptive field size, which demands the large parameter size and computational cost. [53] employs time-depth separable convolution to reduce the computation and memory overhead. For online speech recognition, the intermediate results should be kept for accurate results. Therefore, convolutions with large receptive field size requires a large amount of intermediate buffer, which is inevitably much less efficient for on-device streaming speech recognition.

In [65], they discuss how convolutional neural networks (CNNs) find out positional information. This work shows that CNNs with a large receptive field inherently learns the positional information. Additionally, this work shows that the recognition result heavily depends on this positional information. They show that a sufficient receptive field size and zero-padding are required for the convolutional models to learn positional information.

From the early work of convolutional sequence-to-sequence models [24] for ma-

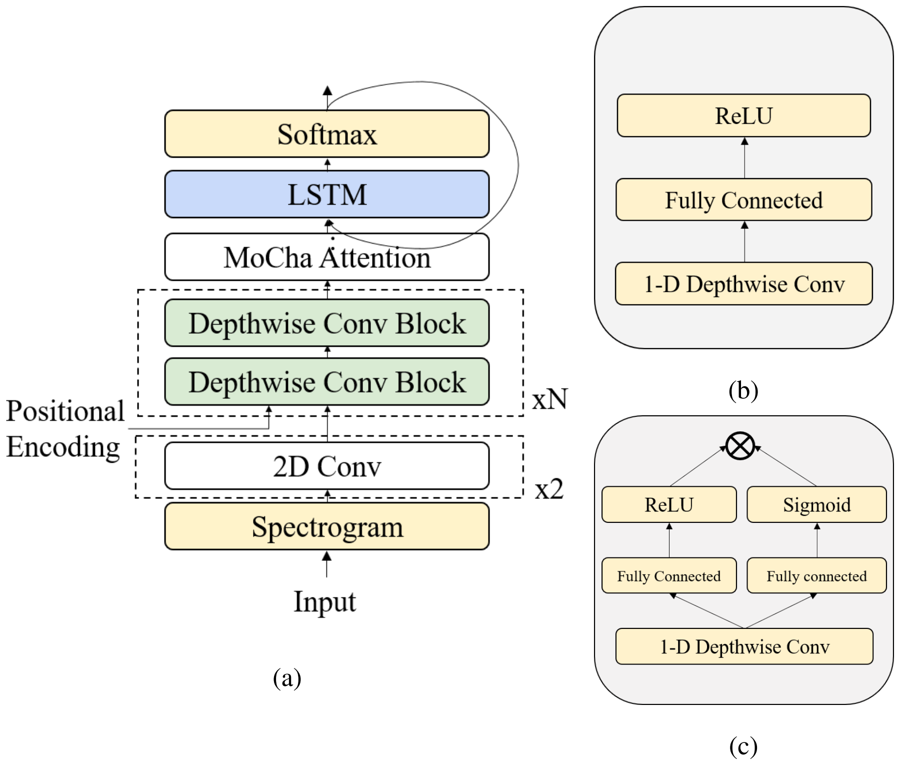


Figure 5.1: (a) The attention-based model with 1-D depthwise convolutions and positional information for encoder. (b) A depthwise convolution block. (c) A block with gating structure.

chine translation, positional embedding vectors are often trained along with the input word embedding. However, the importance of positional information is often overlooked for the convolutional models for speech recognition tasks. For example, [53] does not include positional embedding for the encoder. It mandates the large filter size for wider context, additional training and decoding strategy to avoid misalignment in attention.

5.3 End-to-End Model for Speech Recognition

5.3.1 Model description

The attention-based speech recognition model is based on [45], except that 1-D depth-wise convolutional layers are used for the encoder. Each layer is computed as follows:

$$\begin{aligned} x'_{t,k} &= \sum_{i=-(T-1)/2}^{(T-1)/2} W_{i,k} \cdot x_{t+i,k} \\ \mathbf{h}_t &= f(\mathbf{V}\mathbf{x}'_t + \mathbf{b}), \end{aligned} \tag{5.1}$$

where f is the activation function, and $\mathbf{W} \in \mathbb{R}^{T\times D}, \mathbf{V} \in \mathbb{R}^{D\times D}$ are trainable variables. T and D denote the width and output dimensions of a convolution, respectively. For the encoder, we apply two 2-D convolutions to the input features in the frequency- and time-axis. The 1-D depth-wise convolutional layers are stacked on top of the 2-D convolutions as shown in Figure 5.1. We employ a residual connection for every two convolutional layers. Layer normalization [67] is applied after residual connections for better convergence.

In the conventional global soft attention model, the attention weights $\alpha_{i,t}$ and energies $e_{i,t}$ for the encoder time-step t and decoder step i are obtained by:

$$\begin{aligned} e_{i,t} &= \mathbf{v}_e^T \cdot \tanh(\mathbf{W}[\mathbf{s}_i, \mathbf{h}_t, \beta_{i,t}]) \\ \boldsymbol{\alpha}_i &= \text{softmax}(\mathbf{e}_i), \end{aligned} \tag{5.2}$$

where $\mathbf{v}$ is a trainable vector, $\mathbf{W}$ is a trainable matrix, $\mathbf{s}_i$ the current decoder state, and $\mathbf{h}_t$ denotes the output of the last layer of the encoder. $\beta_{i,t}$ is the attention weight feedback which is defined as:

$$\beta_{i,t} = \sigma(\mathbf{v}_\beta^T \mathbf{h}_t) \sum_{k=1}^{i-1} \alpha_{k,t} \tag{5.3}$$

The attention context vector is given as:

$$\mathbf{c}_i = \sum_t \alpha_{i,t} \mathbf{h}_t. \tag{5.4}$$

The decoder is a single-layer long short-term memory [9] (LSTM) that is computed as follows:

$$\mathbf{s}_i = \text{LSTM}(\mathbf{s}_{i-1}, [\mathbf{y}_{i-1}, \mathbf{c}_{i-1}]). \tag{5.5}$$

5.3.2 Monotonic chunkwise attention

The soft global attention in (5.2) requires the entire input sequence before generating the first output label. This characteristic is not suitable for *streaming* inference in speech recognition. To overcome this problem, several local attention approaches have been proposed [84, 85, 86]. One of the widely used approaches is Monotonic Chunkwise Attention (MoCha) [29]. The attention in Mocha consists of monotonic attention and chunkwise attenton, where monotonic attention determines the location for the decoder to attend, and the encoder outputs in the location are summed with weights obtained by chunkwise attention. The monotonic attention approach is described as follows:

$$\begin{aligned} e_{i,t} &= \mathbf{v}_m^T \cdot \tanh(\mathbf{W}_m[\mathbf{s}_i, \mathbf{h}_t]) \\ \boldsymbol{p}_i &= \sigma(\mathbf{e}_i) \\ \boldsymbol{z}_i &\sim \text{Bernoulli}(\boldsymbol{p}_i) \end{aligned} \tag{5.6}$$

The monotonic attention location is determined as t_i which is the smallest j satisfying $z_{i,j} = 1$ and $j \geq t_{i-1}$. After t_i is obtained from the monotonic attention, chunkwise attention with chunk size C is computed as follows:

$$\begin{aligned} u_{i,t} &= \mathbf{v}_c^T \cdot \tanh(\mathbf{W}_c[\mathbf{s}_i, \mathbf{h}_t]) \\ \alpha_{i,t} &= \frac{\exp(u_{i,t})}{\sum_{t=t_i-C}^{t_i} \exp(u_{i,t})}, \quad t_i - C \leq t \leq t_i \\ \mathbf{c}_i &= \sum_{t=t_i-C}^{t_i} \alpha_{i,t}\mathbf{h}_t. \end{aligned} \tag{5.7}$$

The sampling process of the monotonic attention is non-differentiable therefore cannot be trained with back-propagation. Thus, the expected value of $\mathbf{c}_i$ is used instead during the training. Presumably due to this discrepancy between the training and the inference, it is usually challenging to train a high performance MoCha model. The training of MoCha models is usually sensitive to the hyper-parameter settings such as the learning rate and the gradient clipping and easily diverges if the model size is large or training data is not enough. Therefore, we first train the global attention model, which is more stable. After the global-attention model is converged, the weights of the encoder part of the MoCha model are initialized from the this global attention model.

5.3.3 Positional encoding

In the Transformer-based models [28], the positional encoding vector is usually added to the input vector to the encoder. When these transformer-based models are employed for speech recognition, a linear transformation is often applied to the positional encoding before this addition step [58]. In this work, we employ the positional encoding vector as proposed in [28]:

$$\begin{aligned} pe_{i,2k} &= \sin(i/10000^{(2k)/D}) \\ pe_{i,2k+1} &= \cos(i/10000^{(2k+1)/D}) \end{aligned} \tag{5.8}$$

where i and D denote the decoder time step and the hidden dimension of the layer, respectively. In our experiments, the best performance is achieved when this positional encoding vector is concatenated to the output of 2D convolutions. In the meanwhile, when this positional encoding vector is added or concatenated to the output of the convolutional encoders, the model parameters diverge during the training phase.

Trainable positional embedding vectors are often applied to convolutional models [24] for natural language processing (NLP) tasks. However, this approach is not applicable to speech recognition tasks since the input sequences in speech recognition are usually much longer that those in NLP tasks. Additionally, the variation of this input sequence length is significantly larger in speech recognition.

5.4 Experimental Results

The experiments were performed using the RETURNN framework [68]. TED-LIUM release 2 [69] was used for training, which contains 200 hours of speech. We followed the data preprocessing pipeline used in [70]. We use a 40-dimensional mel-frequency cepstral coefficient (MFCC) as the input feature, which is extracted every 10 ms with a 25 ms window size. Byte-pair encoding (BPE) [71] with a vocabulary size of 1K was used for the output labels. The layer-wise pretraining is only applied to LSTM-based models because it lowers the accuracy when used for convolutional models. The decoder was a single-layer 1000-dimensional LSTM algorithm for all models. Joint CTC training [87] is applied when training the global-attention model. The initial learning rate was 8e-3 and decayed with factor of 0.8 if the validation loss is not decreased for 4 epochs.

Table 5.2: TED-LIUM release 2 results of the models with different filter size. Monotonic chunkwise attention is applied for all models. Pos. denotes that positional encoding is applied.

Model	**WER [%]**	
	dev	test
Conv. 15x2048 (T=3)	18.40	17.57
Conv. 15x2048 (T=5)	17.22	15.62
Conv. 15x2048 (T=7)	16.54	15.10
Conv. 15x2048 (T=11)	14.76	13.28
Conv. 15x2048 (T=3) + Pos.	15.41	13.77
Conv. 15x2048 (T=5) + Pos.	14.10	12.80
Conv. 15x2048 (T=7) + Pos.	15.02	13.18
Conv. 15x2048 (T=11) + Pos.	14.57	13.06

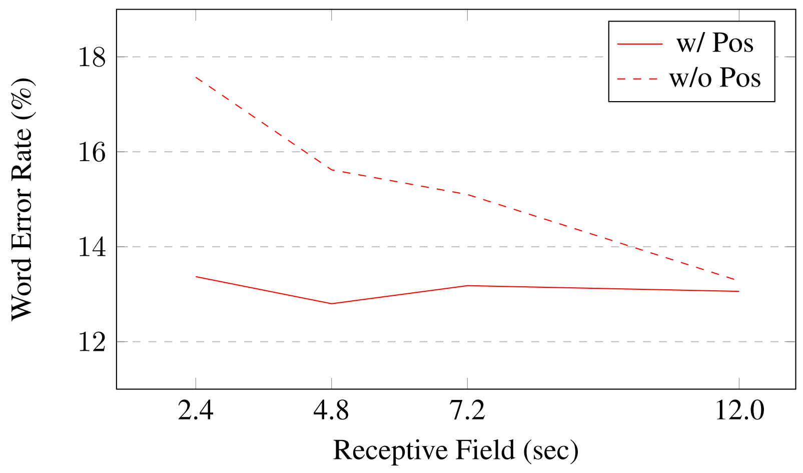


Figure 5.2: Test WER on TED-LIUMv2 comparing models with different receptive field sizes.

The experimental results with the convolutional models on TED-LIUMv2 are shown in Table 5.2. A 15x2048 model denotes that 15 convolutional blocks with $D = 2048$ are stacked. We trained the 15x2048 convolutional model while changing the filter width of depthwise convolutions. We applied max-pooling with a size of 2 for the initial three convolutional layers. All the models have approximately 84M parameters regardless of the filter width because the number of parameters for depth-wise convolution is very small. The results show that positional encoding improved the accuracy of models consistently, especially for those with small filter sizes. Fig. 5.2 shows the word error rate (WER) of the models with different receptive field sizes. The length of the receptive field is calculated by $(T - 1) \times 80ms \times \#layers$, where T is the filter width. The WER of the models without positional encoding sharply increases when the receptive field length is reduced. By comparison, when positional encoding is employed, the WER is not significantly affected by the length of the receptive field. This result clearly demonstrates the effectiveness of positional encoding on CNN-based models.

Transcription 'DanielKahneman_2010_399.58_417.69.ogg'

... WHAT DEFINES A STORY ARE CHANGES SIGNIFICANT MOMENTS **AND** ENDINGS ENDINGS ARE VERY VERY IMPORTANT **AND** IN THIS CASE YOU KNOW THE ENDING DOMINATED

Without positional encoding

... WHAT DEFINES A STORY ARE CHANGES SIGNIFICANT MOMENTS **AND** IN THIS CASE THE ENDING DOMINATED

With positional encoding

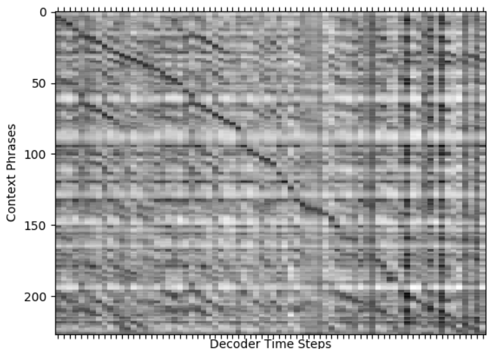
... WHAT DEFINES A STORY ARE CHANGES SIGNIFICANT MOMENTS **AND** ENDINGS AND THINGS OF VERY VERY IMPORTANT **AND** IN THIS CASE THE ENDING DOMINATED

Figure 5.3: The original transcript and the decoded results with and without the positional encoding.

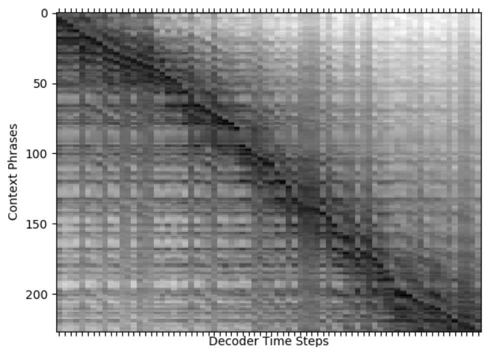
5.4.1 Effect of positional encoding

The decoding results show that *looping* and *skipping* frequently happen with the model without the positional encoding. Figure 5.3 shows the example where skipping occurs only in the model without positional encoding but not in the one with positional encoding. Figure 5.4 visualize the attention weight for the example. Monotonic attention energy $\mathbf{e}_i$ in Eq. (5.6) and attention weight $\alpha_{i,t}$ in Eq. (5.7) are plotted in Figure 5.4. We plotted $\mathbf{e}_i$ with the teacher forcing as in the training, while $\alpha_{i,t}$ is plotted when the previous time-step output is given to the decoder as in the inference. Note that $\mathbf{e}_i$ has the most critical effect on determining where to attend in the encoder outputs. In Fig. 5.4 (a) the attention energy has a high value in the area indicated by the red box. This induces the skipping of attention in the inference time as shown in Fig. 5.4 (b). On the other hand, The attention energy is more concentrated around the diagonal components in Fig. 5.4 (c). This suggest that the location is considered for the attention when the positional encoding is given to the input.

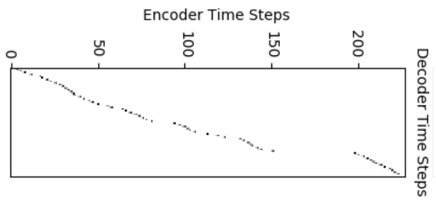
To identify the effect of the positional encoding, we applied the principal component analysis (PCA) to the output of encoder [72]. Figure 5.5 shows the PCA result of the model with and without the positional encoding. We obtain this scatter plot using



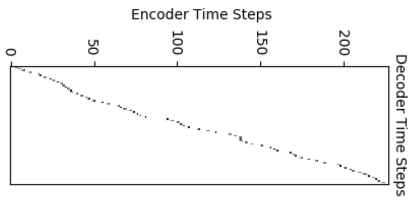
(a)



(b)

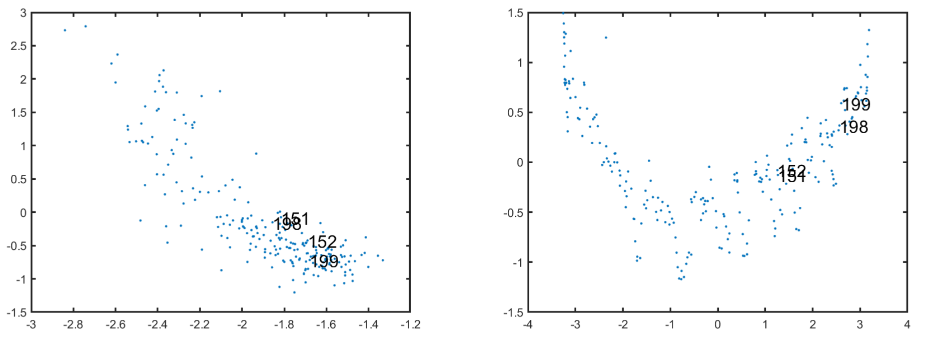


(c)



(d)

Figure 5.4: Attention energy $\mathbf{e}_i$ **(up)** and weight $\boldsymbol{\alpha}_i$ **(down)** of models (a) (c) without and (b) (d) with positional encoding are shown. Darker pixels indicate higher values. 'DanielKahneman_2010_399.58_417.69.ogg' in the test set is used for plotting.



(a) (b)

Figure 5.5: The visualization of encoder output using PCA. The first two principal components are used for visualization. The points of 151-152th and 198-199th steps are indicated with text, which correspond to pronunciation of the word ‘and’. (a) filter width = 3 (b) filter width = 3 with positional encoding.

the first two principal components. Each point in the figure corresponds to the single time-step output of the encoder. Specifically, we analyze the data obtained from the 151-152th and the 198-199th time steps, where the skipping happens. These locations correspond to the first and second pronunciation of ‘and’, respectively. In Figure 5.5 (a), encoder outputs contain similar components for repeated words. When the positional encoding is applied, the outputs are more discriminative as shown in Figure 5.5 (b).

5.4.2 Comparison with other models

Table 5.3 shows the WER of the models with global soft attention and monotonic chunkwise attention. The model with MoCha has 0.22% higher WER than the one with global soft attention. The chunk size of more than two does not show any improvement in terms of WER.

Table 5.4 compares the results of the convolutional models with other structures. The results with LSTM- and self-attention-based models are included for comparison. Unidirectional LSTM has higher WERs than convolutional models with a comparable

Table 5.3: TED-LIUM release 2 results of the models with different attention algorithm.

Model	**WER [%]**	
	dev	test
Conv. 15x2048 (T=5) + (Global Soft Attention)	18.06	15.17
Conv. 15x2048 (T=5) + Pos. (Global Soft Attention)	15.24	12.58
Conv. 15x2048 (T=5) + Pos. (Chunk Size = 2)	14.10	12.80
Conv. 15x2048 (T=5) + Pos. (Chunk Size = 4)	14.17	12.88

Table 5.4: Experimental results with convolutional models of different sizes. LSTM and Transformer-based models are included for comparison. For all the models in this table, we use the same decoder consisting of a single LSTM layer with a 1000 unit size.

Model	**WER [%]**		**Params.**
	dev	test	
Global Soft Attention			
Transformer [70]	11.7	12.5	100M
Bidirectional LSTM 6x1024 [70]	11.7	10.5	161M
Unidirectional LSTM 6x1536	16.78	14.42	127M
Conv. 15x2048 (T=5) + Pos.	15.24	12.58	84M
Gated Conv. 35x2048 (T=5) + Pos.	12.81	10.60	313M
MoCha			
Unidirectional LSTM 6x1536	19.30	17.88	127M
Conv. 15x2048 (T=5) + Pos.	14.10	12.80	84M
Gated Conv. 35x2048 (T=5) + Pos.	12.96	11.20	313M

Table 5.5: Execution time of encoder for 1 second of speech according to the computation chunk size.

Model	1sec	0.1sec
LSTM 6x1536	2.07	2.07
Conv. 15x2048 (T = 15)	0.751	1.89
Conv. 15x2048 (T = 3)	0.727	1.75

number of parameters. The proposed method can be applied to other convolutional structures. We tried gated convolution [27], which has been successfully applied to speech recognition tasks. The experimental results show that using positional encoding improves the accuracy of deeper structures. Conv. 15x2048 (T=5) with MoCha models shows competitive accuracy when compared to global-attention model with a comparable model size. With deeper structure with gated convolution, we obtained a streaming speech recognition model with a WER of 11.20% on the TED-LIUM v2 test set.

5.4.3 Execution time analysis

The computation time for the encoder is presented in Table 5.5. Real-time factor (RTF) is measured on Samsung Galaxy S6 device. The system has 2.1GHz ARM Cortex-A57 CPU. TensorFlow Lite Android benchmark tool is used to measure the computation time. The processing time can be reduced if the computation chunk size increases because the parallelization factor increases and the number of memory access reduces. The single stream of input is processed for the measurement.

5.5 Concluding Remarks

In this chapter, we introduce a new end-to-end streaming speech recognition model employing the convolutional encoder. It has been frequently observed that convolutional models with small filter sizes lack the ability to identify the positional information. This problem is considered to be the primary reason of the looping or skipping problems in the attention-based end-to-end speech recognition. By explicitly adding the positional encoding, we successfully resolve these looping and skipping problems inherent with CNN models with small receptive fields. This proposed method does not require any modifications to the existing model structures nor the training strategies. Furthermore, this approach does not incur any noticeable computational overhead. Experimental results demonstrate that our algorithm has significant advantages over other approaches based on Transformers or RNNs in terms of WERs, the parameter size, and the computational cost.

Chapter 6

Conclusion

In this dissertation, we discussed the implementation of an on-device neural network based speech recognition system. Several parallelizable neural network architectures were applied and the efficient training method for the models was proposed.

In Chapter 2, the SRU-based ASR system was developed. SRU has much lower accuracy than LSTM-based models, but achieved a competitive accuracy when combined with depthwise convolution. The depthwise convolution required a little computational overhead. The beam search decoding is applied with RNN-based LM. The entire system has 15MB parameter sizes and runs real-time in a single core CPU. The system has comparable recognition performance to offline speech recognition model.

An on-device streaming ASR system using depthwise convolution-based models was proposed in Chapter 3. The memory bandwidth requirements were reduced by parameter quantization and the multi-time step parallelization technique. The developed model operates in 0.2 RTF with a 900MHz CPU using only 1.2MB memory footprint for parameters. The system can be applied when the computation and power budgets are scarce, such as in keyword spotting and always-on speech recognition.

In chapter 4, it was demonstrated that the convolutional models with small filter sizes lack the ability to identify positional information. Convolution with a small filter size incurs looping or skipping problems in end-to-end speech recognition. By adding explicit positional encoding, we prevented severe performance degradation of models with small receptive fields. Since convolutional encoders support fast training and inference, the proposed method is suitable for developing an on-device low-power speech recognition system.

In Chapter 5, an end-to-end streaming attention-based speech recognition model was developed by employing the method proposed in Chapter 4. The proposed method did not add any noticeable computational overhead, without requiring modification in the training procedure. Experimental results indicated that the proposed algorithm has significant advantages over other approaches, such as Transformers or RNNs, in terms of WERs, the parameter size, and the computational cost. The developed model, with the parameter size of about 100M, can operate in real-time with a single core ARM CPU for smartphones, and the inference time is only about one third of the LSTM RNN-based attention model.

Bibliography

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

[2] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *International Conference on Machine Learning (ICML)*, 2016, pp. 173–182.

[3] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina *et al.*, "State-of-the-art speech recognition with sequence-to-sequence models," in *Acoustics, Speech and Signal Processing (ICASSP), 2018 IEEE International Conference on.* IEEE, 2018, pp. 4774–4778.

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

[5] K. Rao, H. Sak, and R. Prabhavalkar, "Exploring architectures, data and units for streaming end-to-end speech recognition with RNN-transducer," in *Automatic Speech Recognition and Understanding Workshop (ASRU), 2017 IEEE.* IEEE, 2017, pp. 193–199.

[6] E. Battenberg, J. Chen, R. Child, A. Coates, Y. G. Y. Li, H. Liu, S. Satheesh, A. Sriram, and Z. Zhu, "Exploring neural transducers for end-to-end speech recognition," in *Automatic Speech Recognition and Understanding Workshop (ASRU), 2017 IEEE.* IEEE, 2017, pp. 206–213.

[7] C.-F. Yeh, J. Mahadeokar, K. Kalgaonkar, Y. Wang, D. Le, M. Jain, K. Schubert, C. Fuegen, and M. L. Seltzer, "Transformer-transducer: End-to-end speech recognition with self-attention," *arXiv preprint arXiv:1910.12977*, 2019.

[8] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *Proc. Interspeech 2019*, pp. 2613–2617, 2019.

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

[10] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *Syntax, Semantics and Structure in Statistical Translation*, p. 103, 2014.

[11] C. Xu, J. Yao, Z. Lin, W. Ou, Y. Cao, Z. Wang, and H. Zha, "Alternating multi-bit quantization for recurrent neural networks," *International Conference on Learning Representations (ICLR)*, 2018.

[12] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "Fpga-based low-power speech recognition with recurrent neural networks," in *2016 IEEE International Workshop on Signal Processing Systems (SiPS).* IEEE, 2016, pp. 230–235.

[13] S. Kapur, A. Mishra, and D. Marr, "Low precision rnns: Quantizing rnns without losing accuracy," *arXiv preprint arXiv:1710.07706*, 2017.

[14] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, “Exploring sparsity in recurrent neural networks,” *International Conference on Learning Representations (ICLR)*, 2017.

[15] M. S. Zhang and B. Stadie, “One-shot pruning of recurrent neural networks by jacobian spectrum evaluation,” *arXiv preprint arXiv:1912.00120*, 2019.

[16] Z. Wang, J. Lin, and Z. Wang, “Hardware-oriented compression of long short-term memory for efficient inference,” *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 984–988, 2018.

[17] R. Prabhavalkar, O. Alsharif, A. Bruguier, and I. McGraw, “On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition,” in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5970–5974.

[18] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, “ESE: Efficient speech recognition engine with sparse LSTM on FPGA,” in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 75–84.

[19] D. Balduzzi and M. Ghifary, “Strongly-typed recurrent neural networks,” *arXiv preprint arXiv:1602.02218*, 2016.

[20] J. Bradbury, S. Merity, C. Xiong, and R. Socher, “Quasi-recurrent neural networks,” *International Conference on Learning Representations (ICLR)*, 2017.

[21] T. Lei and Y. Zhang, “Training RNNs as fast as CNNs,” *arXiv preprint arXiv:1709.02755*, 2017.

[22] E. Martin and C. Cundy, “Parallelizing linear recurrent neural nets over sequence length,” *International Conference on Learning Representations (ICLR)*, 2018.

[23] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, “Neural machine translation in linear time,” *arXiv preprint arXiv:1610.10099v2*, 2017.

[24] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional sequence to sequence learning,” in *International Conference on Machine Learning (ICML)*, 2017, pp. 1243–1252.

[25] F. Wu, A. Fan, A. Baevski, Y. Dauphin, and M. Auli, “Pay less attention with lightweight and dynamic convolutions,” *International Conference on Learning Representations (ICLR)*, 2019.

[26] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” in *International Conference on Machine Learning (ICML)*, 2017, pp. 933–941.

[27] V. Liptchinsky, G. Synnaeve, and R. Collobert, “Letter-based speech recognition with gated ConvNets,” *arXiv preprint arXiv:1712.09444*, 2017.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 6000–6010.

[29] C.-C. Chiu and C. Raffel, “Monotonic chunkwise attention,” *International Conference on Learning Representations (ICLR)*, 2018.

[30] J. Park, Y. Boo, I. Choi, S. Shin, and W. Sung, “Fully neural network based speech recognition on mobile and embedded devices,” in *Advances in Neural Information Processing Systems*, 2018, pp. 10 620–10 630.

[31] J. Park, X. Qian, Y. Jo, and W. Sung, “Low-latency lightweight streaming speech recognition with 8-bit quantized simple gated convolutional neural networks,” in

2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 05 2020, pp. 1803–1807.

[32] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky, "Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices," in *Acoustics, Speech and Signal Processing (ICASSP), 2006 IEEE International Conference on*. IEEE, 2006.

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

[34] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[35] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2006, pp. 369–376.

[36] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4580–4584.

[37] K. Hwang and W. Sung, "Character-level incremental speech recognition with recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5335–5339.

[38] ——, “Character-level language modeling with hierarchical recurrent neural networks,” in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 5720–5724.

[39] Z. Chen, Y. Zhuang, Y. Qian, K. Yu, Z. Chen, Y. Zhuang, Y. Qian, K. Yu, K. Yu, Y. Zhuang *et al.*, “Phone synchronous speech recognition with CTC lattices,” *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 25, no. 1, pp. 90–101, 2017.

[40] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[41] Y. Gal and Z. Ghahramani, “A theoretically grounded application of dropout in recurrent neural networks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 1019–1027.

[42] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[43] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[44] Y. Zhou, C. Xiong, and R. Socher, “Improving end-to-end speech recognition with policy learning,” in *Acoustics, Speech and Signal Processing (ICASSP), 2018 IEEE International Conference on*. IEEE, 2018, pp. 5819–5823.

[45] A. Zeyer, K. Irie, R. Schlüter, and H. Ney, “Improved training of end-to-end attention models for speech recognition,” *Interspeech*, 2018.

[46] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “Librispeech: an ASR corpus based on public domain audio books,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5206–5210.

[47] Z. Xianyi, W. Qian, and Z. Chothia, “Openblas,” *URL: http://xianyi. github. io/OpenBLAS*.

[48] “Gemmlowp: A small self-contained low-precision gemm library,” *URL: https://github. com/google/gemmlowp*.

[49] C. Lüscher, E. Beck, K. Irie, M. Kitza, W. Michel, A. Zeyer, R. Schlüter, and H. Ney, “Rwth asr systems for librispeech: Hybrid vs attention,” 2019.

[50] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang *et al.*, “Streaming end-to-end speech recognition for mobile devices,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6381–6385.

[51] J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. T. Gadde, “Jasper: An end-to-end convolutional neural acoustic model,” in *Proceedings of Interspeech*, 2019.

[52] L. Lee, J. Park, and W. Sung, “Simple gated convnet for small footprint acoustic modeling,” in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019.

[53] A. Hannun, A. Lee, Q. Xu, and R. Collobert, “Sequence-to-sequence speech recognition with time-depth separable convolutions,” in *Proceedings of Interspeech*, 2019.

[54] W. Wen, Y. Wang, F. Yan, C. Xu, C. Wu, Y. Chen, and H. Li, “SmoothOut: Smoothing out sharp minima to improve generalization in deep learning,” *arXiv preprint arXiv:1805.07898*, 2018.

[55] A. Graves, “Practical variational inference for neural networks,” in *Advances in neural information processing systems*, 2011, pp. 2348–2356.

[56] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[57] 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.

[58] N.-Q. Pham, T.-S. Nguyen, J. Niehues, M. Müller, and A. Waibel, “Very deep self-attention networks for end-to-end speech recognition,” *Proc. Interspeech 2019*, pp. 66–70, 2019.

[59] V. Pratap, Q. Xu, J. Kahn, G. Avidov, T. Likhomanenko, A. Hannun, V. Liptchinsky, G. Synnaeve, and R. Collobert, “Scaling up online speech recognition using convnets,” 2020.

[60] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[61] K. Kim, K. Lee, D. Gowda, J. Park, S. Kim, S. Jin, Y.-Y. Lee, J. Yeo, D. Kim, S. Jung *et al.*, “Attention based on-device streaming speech recognition with large speech corpus,” in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 956–963.

[62] J. Chorowski and N. Jaitly, “Towards better decoding and language model integration in sequence to sequence models,” *Proc. Interspeech 2017*, pp. 523–527, 2017.

[63] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6124–6128.

[64] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in neural information processing systems*, 2015, pp. 577–585.

[65] M. A. Islam, S. Jia, and N. D. Bruce, "How much position information do convolutional neural networks encode?" *International Conference on Learning Representations (ICLR)*, 2020.

[66] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li, "Modeling coverage for neural machine translation," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 76–85.

[67] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[68] P. Doetsch, A. Zeyer, P. Voigtlaender, I. Kulikov, R. Schlüter, and H. Ney, "Returnn: The rwth extensible training framework for universal recurrent neural networks," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5345–5349.

[69] A. Rousseau, P. Deléglise, and Y. Esteve, "Enhancing the TED-LIUM corpus with selected data for language modeling and more ted talks." in *LREC*, 2014, pp. 3935–3939.

[70] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney, "A comparison of transformer and LSTM encoder decoder models for ASR," in *IEEE Automatic Speech Recognition and Understanding Workshop, Sentosa, Singapore*, 2019.

[71] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1715–1725.

[72] H. Hotelling, “Analysis of a complex of statistical variables into principal components.” *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[73] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[74] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi, “Simple recurrent units for highly parallelizable recurrence,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 4470–4481.

[75] D. Povey, H. Hadian, P. Ghahremani, K. Li, and S. Khudanpur, “A time-restricted self-attention layer for asr,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2018, pp. 5874–5878.

[76] N. Moritz, T. Hori, and J. Le, “Streaming automatic speech recognition with the transformer model,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6074–6078.

[77] T. N. Sainath, Y. He, B. Li, A. Narayanan, R. Pang, A. Bruguier, S.-y. Chang, W. Li, R. Alvarez, Z. Chen *et al.*, “A streaming on-device end-to-end model surpassing server-side conventional model quality and latency,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2020, pp. 6059–6063.

[78] B. Liu, S. Cao, S. Sun, W. Zhang, and L. Ma, “Multi-head monotonic chunkwise attention for online speech recognition,” *arXiv preprint arXiv:2005.00205*, 2020.

[79] S. Xue and Z. Yan, "Improving latency-controlled blstm acoustic models for online speech recognition," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2017, pp. 5340–5344.

[80] Y. Wang, A. Mohamed, D. Le, C. Liu, A. Xiao, J. Mahadeokar, H. Huang, A. Tjandra, X. Zhang, F. Zhang *et al.*, "Transformer-based acoustic modeling for hybrid speech recognition," *arXiv preprint arXiv:1910.09799*, 2019.

[81] J. Salazar, K. Kirchhoff, and Z. Huang, "Self-attention networks for connectionist temporal classification in speech recognition," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2019, pp. 7115–7119.

[82] L. Dong, S. Xu, and B. Xu, "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2018, pp. 5884–5888.

[83] K. J. Han, J. Huang, Y. Tang, X. He, and B. Zhou, "Multi-stride self-attention for speech recognition," *Proc. Interspeech 2019*, pp. 2788–2792, 2019.

[84] C. Raffel, M. Luong, P. J. Liu, R. J. Weiss, and D. Eck, "Online and linear-time attention by enforcing monotonic alignments," in *ICML'17: Proceedings of the 34th International Conference on Machine Learning*, vol. 70, Aug. 2017, p. 2837–2846.

[85] A. Tjandra, S. Sakti, and S. Nakamura, "Local monotonic attention mechanism for end-to-end speech and language processing," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2017, pp. 431–440.

[86] J. Hou, S. Zhang, and L.-R. Dai, "Gaussian prediction based attention for online end-to-end speech recognition." in *INTERSPEECH*, 2017, pp. 3692–3696.

[87] S. Kim, T. Hori, and S. Watanabe, “Joint ctc-attention based end-to-end speech recognition using multi-task learning,” in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2017, pp. 4835–4839.

초록

최근 모바일 및 임베디드 기기에서 실시간 동작하는 음성 인식 시스템을 개발하는 것이 큰 관심을 받고 있다. 깊은 인공 신경망 음성인식은 많은 양의 연산을 필요로 하는 반면, 모바일 기기의 메모리 대역폭이나 전력은 제한되어 있다. 이러한 한계 때문에 서버 기반 구현이 보통 사용되어지지만, 이는 지연 시간 및 사생활 침해 문제를 일으킨다. 따라서 모바일 기기 상 동작하는 음성 인식 시스템의 요구가 커지고 있다. 음성 인식 시스템에 주로 사용되는 모델은 재귀형 인공 신경망이다. 재귀형 인공 신경망의 모델 크기는 보통 캐시의 크기보다 크고 피드백 구조 때문에 재사용이 어렵기 때문에 많은 DRAM 접근을 필요로 한다. 이러한 문제를 해결하기 위해 다중 시간의 입력에대해 병렬화 가능한 모델을 이용한 음성 인식 시스템을 제안한다. 다중 시간 병렬화 기법은 한 번의 메모리 접근으로 여러 시간의 출력을 동시에 계산하는 방법이다. 병렬화 수에 따라 DRAM 접근 횟수를 줄일 수 있기 때문에, 병렬화 가능한 모델에 대하여 빠른 연산이 가능하다.

단순 재귀 유닛과 1차원 컨벌루션을 이용한 CTC 모델을 제시하였다. 문자와 단어 조각 수준의 모델이 개발되었다. 각 출력 단위에 해당하는 재귀형 신경망 기반 언어 모델을 이용하여 디코딩에 사용되었다. 전체 15MB의 메모리 크기로 WSJ 에서 높은 수준의 인식 성능을 얻었으며 GPU나 기타 하드웨어 없이 1개의 ARM CPU 코어로 실시간 처리를 달성하였다.

또한 단순 컨벌루션 인공 신경망 (SGCN)을 이용한 낮은 지연시간을 가지는 음성인식 시스템을 개발하였다. SGCN은 1M의 매우 낮은 변수 갯수로도 경쟁력 있는 인식 정확도를 보여준다. 추가적으로 8-bit 양자화를 적용하여 메모리 크기와 연산

시간을 감소 시켰다. 해당 시스템은 0.4초의 이론적 지연시간을 가지며 900MHz의 CPU 상에서 0.2의 RTF로 동작하였다.

추가적으로, 깊이별 컨벌루션 인코더를 이용한 어텐션 기반 모델이 개발되었다. 컨벌루션 기반의 인코더는 재귀형 인공 신경망 기반 모델보다 빠른 처리 속도를 가진다. 하지만 컨벌루션 모델은 높은 성능을 위해서 큰 입력 범위를 필요로 한다. 이는 모델 크기 및 연산량, 그리고 동작 시 메모리 소모를 증가 시킨다. 작은 크기의 입력 범위를 가지는 컨벌루션 인코더는 출력의 반복이나 생략으로 인하여 높은 오차율을 가진다. 이것은 컨벌루션의 시간 불변성 때문으로 여겨지며, 이 문제를 위치 인코딩 벡터를 이용하여 해결하였다. 위치 정보를 이용하여 작은 크기의 필터를 가지는 컨벌루션 모델의 성능을 높일 수 있음을 보였다. 또한 위치 정보가 가지는 영향을 시각화 하였다. 해당 방법을 단조 어텐션을 이용한 모델에 활용하여 컨벌루션 기반의 스트리밍 가능한 음성 인식 시스템을 개발하였다.

주요어: 인공 신경망, 온라인 음성 인식, 임베디드 기기용 머신 러닝

학번: 2015-22786