Exploring Neural Network Architectures For Acoustic Modeling

by

Yu Zhang

B.S., Shanghai Jiao Tong University (2009)
M.S., Shanghai Jiao Tong University (2012)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Author .....
Department of Electrical Engineering and Computer Science
August 31, 2017

Certified by .....
James R. Glass
Senior Research Scientist
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Exploring Neural Network Architectures For Acoustic Modeling

by

Yu Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Deep neural network (DNN)-based acoustic models (AMs) have significantly improved automatic speech recognition (ASR) on many tasks. However, ASR performance still suffers from speaker and environment variability, especially under low-resource, distant microphone, noisy, and reverberant conditions. The goal of this thesis is to explore novel neural architectures that can effectively improve ASR performance.

In the first part of the thesis, we present a well-engineered, efficient open-source framework to enable the creation of arbitrary neural networks for speech recognition. We first design essential components to simplify the creation of a neural network with recurrent loops. Next, we propose several algorithms to speed up neural network training based on this framework. We demonstrate the flexibility and scalability of the toolkit across different benchmarks.

In the second part of the thesis, we propose several new neural models to reduce ASR word error rates (WERs) using the toolkit we created. First, we formulate a new neural architecture loosely inspired by humans to process low-resource languages. Second, we demonstrate a way to enable very deep neural network models by adding more non-linearities and expressive power while keeping the model optimizable and generalizable. Experimental results demonstrate that our approach outperforms several ASR baselines and model variants, yielding a 10% relative WER gain. Third, we incorporate these techniques into an end-to-end recognition model. We experiment with the Wall Street Journal ASR task and achieve 10.5% WER without any dictionary or language model, an 8.5% absolute improvement over the best published result.

Thesis Supervisor: James R. Glass
Title: Senior Research Scientist

Acknowledgments

First, I want to thank Frank Soong and Qiang Huo for encouraging me to pursue a Ph.D. degree. Without them, I would not had such a great experiences in my life. It is still hard to believe that I am almost done with this journey.

I would like to thank my advisor Dr. Jim Glass, who offered me the chance to be part of the family, MIT spoken language systems (SLS) group. Jim's vision for speech processing has motivated me to explore the research topics investigated in this thesis. He always had patience with all my faults and random ideas, and has given me great freedom to pursue the research that I would like to work on. This thesis would not have been possible without him.

I would like to thank my wonderful thesis committee, Regina Barzilay and Tommi Jaakkola. They gave me invaluable advice and comments on my thesis throughout the whole process. Beyond the committee, I have been very fortunate to have the opportunity to be a TA in NLP class for Regina and Tommi. I learned how to organize a large class (200 people!) and how to formulate an incomplete idea as an elegant math solution.

I am grateful to have worked with Dong Yu and Navdeep Jaitly during my internships in MSR and Google. I learned a lot from Dong about how to transform a research idea into an industrial solution and how a brilliant researcher can also be a very good engineer. I also really learned a lot from Navdeep about deep learning, which has been incredibly useful in my research.

I am extremely fortunate to be a part of the SLS group. Many thanks to Ann, Carrie, Chen, Daniel, Dave, Felix, Hassan, Hongyin, Hung-yi, Jackie, Jennifer, JJ, Leo, Mandy, Marcia, Michael, Mitra, Najim, Patrick, Scott, Sree, Stephen, Tuka, Wei-Ning, William, Xue, Yaodong, and Yonatan, for their help and many interesting conversations during my Ph.D. study. Special thanks to our administrative assistant Marcia Davidson for her help.

I am also extremely fortunate to be able to collaborate with many great people, especially for all the insightful ideas and discussions. Thanks to Jasha and Mike, I

learned a lot from the mini-acoustic model meeting with them at MSR. Many thanks to Guoguo, Ekapol, Wei-Ning, and William for the collaborations directly related to this thesis. My work would not have been possible without them. Thanks to Najim, for various discussions and sharing valuable suggestions in my research. Thanks to Jackie, Hung-yi, and Patrik for insightful discussions at MIT. Thanks to Liang, Xiong, Yanmin, Tianxing, and Tian for awesome collaborations with them at the JHU workshop. Thank to Babelon Team, which consists of BBN, BUT, JHU, LIMSI, and NWU. Our weekly calls were very helpful for my research. Thanks to Stephen, Mandy, and Dave, although we never wrote a paper together, I've really enjoyed and benefited a lot from discussions with them. They also helped proofread all of my papers. Thanks to Tara, who provided a lot of feedback and suggestions on the work presented in my thesis. Thanks to Hori-san and Watanabe-san, I enjoyed discussions with them about Seq2Seq models. Thanks to Yuan and Tao, I will never forget all the discussions with them on the fourth floor of the Stata Center. Thanks to Chiyuan, Tianfan, Xuhong, and Zhengdong, etc. I really enjoyed the deep learning reading group we organized.

A big thank you to my girlfriend Xi for her great love and support. I also dedicate this thesis to my wonderful family: my father Yingxin Zhang and my mother Jianwei Li. Many thanks for their support and patience with me throughout my entire Ph.D. life.

Last but not least, I would like to thank my sponsors, PingAn and IARPA for providing the research funding. All this would not have happened without their generous support.

Bibliographic Note

Portions of this thesis are based on prior peer-reviewed publications. Part of the work presented in Chapter 3 was originally published in [124]. We add more implementation details and experimental comparison in this chapter. Chapter 4 was published in [123] and [125]. Most of the work presented in Chapter 5 was published in [124] and [51]. We include more experiments on different datasets. Chapter 6 was published in [126].

Part of the code in this thesis is available at `https://github.com/yzhang87`.

Contents

List of Figures

List of Tables

1

Introduction

Deep learning and artificial neural network research has grown significantly over the past decade in the fields of ASR, natural language processing and computer vision. Compared to traditional methods, deep learning-based approaches require only limited domain knowledge to reach top performance. This is comes at the cost of requiring much larger datasets and the computational power to process them. In speech recognition, for instance, neural network-based (NN) acoustic models have greatly improved ASR performance over traditional Gaussian mixture models (GMMs) on a variety of tasks [21, 93, 47, 94].

These advances aside, the performance of NN-based acoustic models still suffers from challenges: 1) for rich resource conditions such as good quality recording conditions in English, researchers want to push ASR performance to human parity and 2) conversely, low-resource or noisy, reverberant recording conditions, ASR performance is still far from satisfactory. One key trend that has driven the ASR field towards these directions is exploring useful neural architectures that preserve important information, and provide invariance against unwanted noise. For example, huge improvements have been achieved for ASR from more advanced neural network models such as convolutional neural networks [88, 85, 3, 109] to capture spectral invariance, time-delay neural networks [76] and long short-term memory (LSTM) recurrent neural networks (RNNs) [36, 35, 90] to model the dynamic temporal process in speech. The success of neural methods is contingent on specific operational and architectural choices of the neural models. However, the architectural decisions often

come at the cost of huge computation and lack of insight:

- **Flexible and efficient framework for NN development** A standard "recipe" for building a state-of-the-art speech recognizer typically requires thousands of hours of transcribed speech. As a consequence of this huge amount of data, exploring different neural architectures is very time-consuming. For example, in [93], the training time was one month for a standard deep NN on a 2000 hr corpus. Moreover, even such large amounts of data do not exist in some scenarios. Because of task specific traits, implementing a new neural network from scratch can be nontrivial and involve a lot of difficulties. A good framework for development can go a long way forwards alleviating these difficulties.

- **Bringing intuition to the architectural choices of the neural model** Deep learning found success in ASR for three reasons: 1) the representational power comes from speech knowledge, e.g., the hybrid GMM-HMM system; 2) the network is easy to optimize even in a highly non-linear space and 3) the network can be trained easily using large amounts of data. These properties hinge upon architectural choices of the neural model driven by human intuition. For example, some language-specific traits can help us create a better multilingual recognizer [122]. Furthermore, the operational components need to be easily optimized. For example, the LSTM is designed for easy optimization of RNNs which have greatly improved ASR performance [90]. Moreover, computation needs to be affordable to utilize vast quantities speech data.

In this dissertation, I present the efforts I made for a general purpose machine learning toolkit called the computational network toolkit (CNTK) and under this framework, I also explore different neural architecture for various tasks. More specifically, CNTK permits easy architectural and operational variations to explore and to tune neural network structures. It includes not only highly optimized code to speed up training, but also new algorithms to make existing recurrent networks train more efficiently. With an efficient operational platform, I proposed new neural architectures for various tasks. First, I designed a novel system for low-resource language

recognition. Secondly, training such novel networks is not as straightforward as a regular deep neural network. Optimization of such networks has proven to be considerably more difficult due to multiple components in the system. To solve this issue, I approached the problem by using a learned gating mechanism for regulating information flow, which enables the optimization of networks with virtually arbitrary depth. Moreover, we combined all the techniques we developed and other techniques from the vision community, into an end-to-end speech recognition system.

I briefly describe these techniques below.

1.1 Contributions

The primary contributions of this thesis are:

- **Designing a flexible and efficient framework for deep learning** We present a well-engineered, efficient open-source framework that fosters future speech and machine learning research. Besides the engineering effort, my contribution to this framework includes 1) an efficient batching algorithm 2) an efficient training algorithm for recurrent neural networks and 3) building a state-of-the-art benchmark using standard datasets.

- **Building a feedback loop into the acoustic model** We propose a new structured computational network to simulate the prediction, adaptation and correction process of humans for low-resource language recognition. We demonstrate that our new model can improve a strong stacked bottleneck feature based hybrid system. The network itself can further benefit from transfer learning.

- **Easing gradient-based training** We propose a new gated mechanism to train a much deeper network, in order to improve the generalization of an acoustic model. Based on the analysis of the experimental results, we further enhanced performance by adding more controllable gates to the model. We show that this framework can easily train a deeper network without loss of accuracy. We

further demonstrate that our final model could outperform the state-of-the-art on many standard benchmarks.

- **Adding more expressive power and better generalization** We successively trained very deep convolutional networks to add more expressive power and better generalization for end-to-end ASR models. We applied network-in-network principles, batch normalization, residual connections, and convolutional LSTMs to build very deep recurrent and convolutional structures. We experimented with the WSJ ASR task and significantly improved the state-of-the-art system.

1.2 Thesis Overview

The rest of this thesis is organized as follows:

- Chapter 2 reviews concepts related to ASR and DNNs. It also provides descriptions of the corpora and metrics used in the thesis.
- Chapter 3 presents a well-engineered, efficient open-source framework to make it easier to create an arbitrary computational neural network.[1]
- Chapter 4 proposes a feedback mechanism for low-resource languages.
- Chapter 5 presents a new framework to make it easier to use gradient based training and also investigates different variants across different tasks.
- Chapter 6 investigates methods to improve acoustic modeling in an end-to-end scheme.
- Chapter 7 summarizes the key concepts of the thesis, and suggests future work.

[1]This toolkit is joint work with many researchers at Microsoft.

2
Background

2.1 Introduction

This chapter describes some of the building blocks used in this thesis. Section 2.2 first introduces some ASR basics. Section 2.3 details DNN usage for acoustic modeling e.g., CNNs and RNNs. Section 2.4 describes the speech corpus which we use heavily in this thesis.

2.2 Automatic Speech Recognition

ASR is the process of transcribing human speech into text automatically using machines. Figure 2-1 shows the components of a common speech recognition system, which we describe here briefly. An input *audio* recording is first preprocessed using an audio front-end (Feature Extraction) that extracts acoustic features. Most speech recognizers use a frame-based model in which an input waveform is converted into a sequence of *frames* of features of equal duration. Features such as Mel-frequency Cepstral Coecients (MFCC), and Perceptual Linear Prediction (PLP) [22] are examples of this category of features. Speech features have also been developed to use features computed over different length segments [29] or learned by a neural network [86]. Our current body of work deals only with frame-based features. Typically a feature vector is extracted on a small time window of speech, usually on the order of 25 ms. These frames are usually computed every 10 ms.

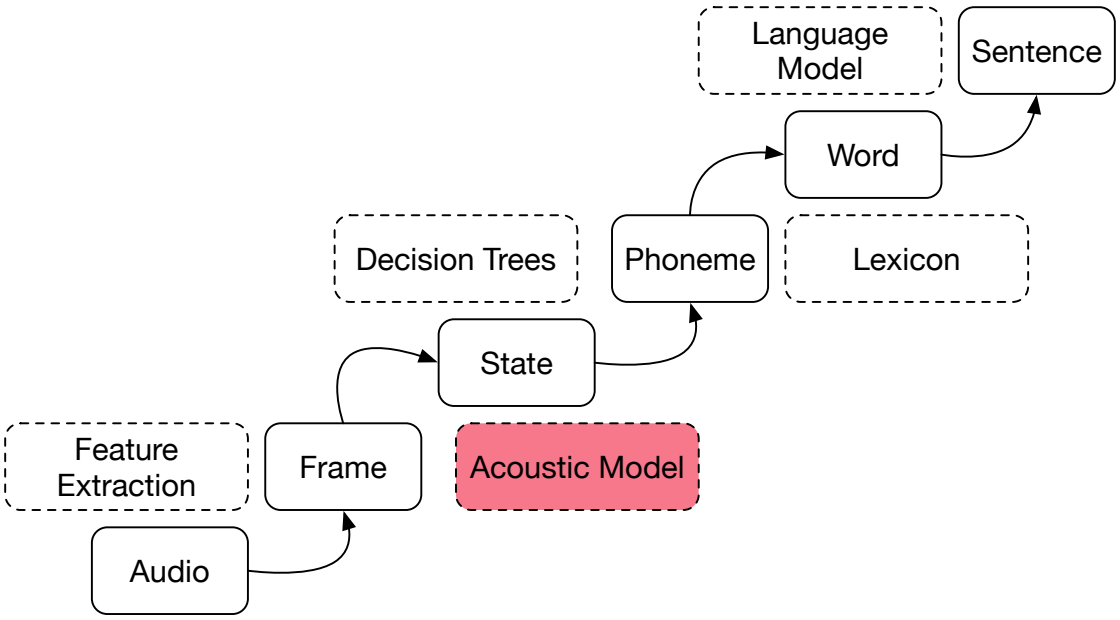


Figure 2-1: A typical speech recognition system. It includes a signal processing frontend, an acoustic model, a pronunciation model, a language model, and a decoder.

The *acoustic model* (the red block in Figure 2-1) is a statistical model of the features that are computed by the acoustic front-end. It is a statistical model of the features conditioned on different spoken sound classes. We denote sound classes as *state* in Figure 2-1 because they are usually represented by states in the Hidden Markov Model (HMM), as described in Section 2.2.1. This model is typically done via a Deep Neural Network (DNN), Convolutional Neural Network (CNN), or RNN acoustic model. Most of work in this thesis attempts to improve the acoustic model.

The *decision tree* maps sub-word units (the hidden state generated by an HMM) to phoneme sequences. Then, the *lexicon*, or *pronunciation model*, maps a sequence of phonemes to a word. The decision tree and pronunciation dictionary are typically fixed, and are rarely updated in the speech pipeline.

The *language model* is a statistical model giving the probability of word sequences independent of the acoustics. The standard Language Models (LMs) used in ASR are n-grams. N-gram models represent the probability of generating the next word given the previous N-1 words. The log probability from the language model is typically linearly combined with the acoustic model score, and then fed into the decoder.

The *decoder* combines the probabilities from the acoustic model and language model to search for the best word sequences under the constraints of the pronunciation model. A brute force search by generating and scoring all candidates one by one is not feasible computationally. Most decoders use a combination of dynamic programming and beam-searching to generate a subset of plausible candidates, and score them at the same time [101, 84]. Modern decoders are usually implemented using Weighted Finite State Transducers (WFSTs) for efficient searching.

We now describe a typical state-of-the-art DNN-HMM speech recognition system with more formal mathematical notation.[1]

2.2.1 Acoustic Model

Given the observation sequence (feature frames), $\mathbf{X}$, extracted from a speech waveform, we want to find the best word sequence $\mathbf{W}^*$ that maximizes the posterior probability $p(\mathbf{W}|\mathbf{X})$

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} p(\mathbf{W}|\mathbf{X}) \tag{2.1}$$

We can decompose this probability into two terms using Bayes's Rule, an *acoustic model*, $p_{\text{AM}}(\mathbf{X}|\mathbf{W})$, and a *language model*, $p_{\text{LM}}(\mathbf{W})$:

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} p_{\text{AM}}(\mathbf{X}|\mathbf{W}).p_{\text{LM}}(\mathbf{W}) \tag{2.2}$$

The first term $p_{\text{AM}}(\mathbf{X}|\mathbf{W})$ in Eq 2.2, the acoustic model, evaluates the likelihood of a speech segment $\mathbf{X}$ being generated by a word sequence $\mathbf{W}$.

In the traditional Gaussian Mixture Model-Hidden Markov Model (GMM-HMM) paradigm, the output probabilities are generated by the GMM, and the sequential property of speech is modeled by the HMM. The hidden states, S, in the HMM form a Markov chain. In this manner, the acoustic feature vectors are generated from hidden states. These hidden states typically represent a subword or phonetic segmentation

[1]We give more details only for the acoustic model, language model, and discriminative training because these are the components related to this thesis. Readers are invited to refer to [84] for more details.

of each word. In other words, we would change Eq 2.2 to:

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \sum_{\mathbf{S}} p(\mathbf{X}, \mathbf{S}|\mathbf{W}) p(\mathbf{W}) \tag{2.3}$$

$$= \arg\max_{\mathbf{W}} \sum_{\mathbf{S}} \prod_{t} p(\mathbf{x}_t|\mathbf{S}, \Omega) p(\mathbf{S}|\mathbf{W}) p(\mathbf{W}) \tag{2.4}$$

where $\mathbf{x}_t, \mathbf{s}_t$ are the observation and hidden state at time t, respectively, and $p(\mathbf{x}_t|\mathbf{S}, \Omega)$ is the acoustic model. Specifically, the parameters Ω are parameterized under the Markovian assumption as:

$$p(\mathbf{X}|\Omega) = \sum_{s} \prod_{t} b_{s_t}(x_t) a_{s_t, s_{t+1}} \tag{2.5}$$

where $b_{s_t}(x_t)$ is the GMM and a_* are the HMM transition probabilities.

In practice, HMM hidden states are typically modeled by 3-state triphones. A triphone is a phone with a left and right context. Each triphone is usually modeled by 3 left-to-right states to handle the transient acoustic dynamics. Systems that model triphones are usually referred to as having context dependent (CD) models, while systems that model just single phonemes without any context use context independent (CI) models.

The parameters of the GMM-HMM acoustic model can be estimated in a Maximum Likelihood (ML) fashion using the Forward-Backward algorithm. At test time, the maximization can be solved by using the Viterbi Algorithm. Readers are invited to refer to [84] for more details.

Recently there has been an explosion of interest in Neural Network models for AM. We will describe the DNN-based AMs in Section 2.3. Also, in this thesis, we will mostly use the DNN-based AM paradigm.

2.2.2 Language Model

The classic Language Model (LM) technique for ASR, $p_{\text{LM}}(W)$ in Eq 2.2 are n-grams [71]. An n-gram language model has the Markovian assumption, typically

conditioning on a 2- to 5-word history, and, therefore, losing long-range context-dependency. Recently, researchers have looked into using RNNs for language modeling as well [73]. However, due to the computational cost, LMs based on RNNs are typically used to re-scoring the N-best lists after the beam search is completed [73]. LMs are usually trained independently from the AMs on large quantity of text data. The sequence-to-sequence model introduced in Chapter 6 attempts to optimize a character-based LM and AM jointly but it usually also needs another word-based LM.

2.2.3 Discriminative Training

In practice, the models trained using the ML criterion do not yield the best results, since the conditional-independence assumptions of the model do not hold for speech data. Many discriminative training methods [77] have been proposed to address this issue. Instead of maximizing the likelihood of the data, discriminative training tries to minimize the confusion in the training data. Commonly used loss functions, are Maximum Mutual Information (MMI), Minimum Phone Error (MPE) and state-level Minimum Bayes Risk (sMBR), which try to minimize the errors in the frame, phone, and state level, respectively. We mainly use sMBR in this thesis for training hybrid DNN-HMM systems, since it often yields the best results [113, 91].

While sequence-discriminative training of neural networks may look trivial at first glance, as it only requires changing the frame-level cross entropy (CE) training criterion to one of the sequence-discriminative training criteria, there are several techniques that are needed to make it work in practice. These include criterion selection, frame-smoothing, language model selection, and so on. The best configuration of these techniques depends on the implementation details [113], as well as the dataset used for training and evaluation, and needs to be optimized when building a state-of-the-art discriminatively trained NN-HMM system. Moreover, if we use more advanced acoustic models such as RNNs, GPU memory is a key constraint. In Chapter 3, we focus on how can we make the sequence training more efficient in the DNN-based AM paradigm.

2.3 Deep Neural Networks for Acoustic Modeling

Over the last decades, there has been an explosion of interest in Neural Network models for AM, LM, and the entire ASR pipeline. DNNs are a particular instance of Neural Networks, where there are many hidden layers. This model was made possible by the recent availability of more powerful hardware such as graphical processing units (GPUs). DNN computations typically require many large matrix multiplications, which can be parallelized easily in the GPU. In ASR, researchers have shown 5-30% relative improvement on various tasks [47]. The power of DNNs is often believed to come from the fact that the representation and the classification are trained jointly. In the following segments, we will discuss the components of the neural network, and how it can be trained and used for acoustic modeling.

2.3.1 DNNs for Acoustic Modeling

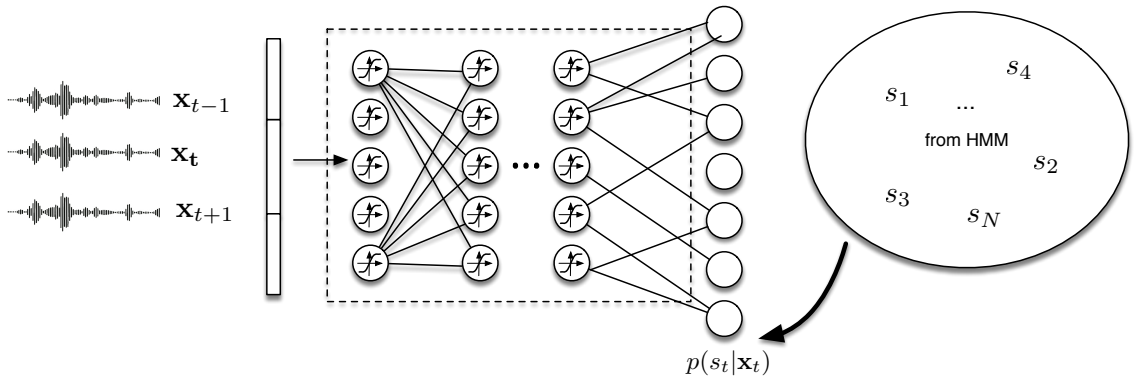


Figure 2-2: The hybrid DNN-HMM approach where the targets are the HMM states and observation probabilities are generated by a DNN. The input $\mathbf{x}_t$ is usually concatenated with adjacent frames to capture more contextual information.

A DNN-based ASR system usually starts with a baseline GMM-HMM speech recognizer that computes frame-level output target labels. This is usually done by force aligning the transcription with the input speech by the GMM-HMM recognizer. This means that the DNN is trying to model the probability of an acoustic frame $\mathbf{x}_t$

being in state posterior s_t at time t [21]:

$$p(s_t|\mathbf{x}_t) = \mathrm{DNN}_{\mathrm{AM}}(\mathbf{x}_t), \tag{2.6}$$

where the DNN can be replaced by any neural network, e.g. RNNs. Since DNNs produce posteriors rather than likelihoods, we need to normalize the DNN outputs using the class priors for decoding:

$$p(\mathbf{x}_t|s_t) \approx \frac{p(s_t|\mathbf{x}_t)}{p(s_t)}. \tag{2.7}$$

This method of using DNN for acoustic modeling in ASR is usually called the "*hybrid DNN-HMM*" as illustrated in Figure 2-2.

The objective function used for hybrid DNN-HMM training is the cross entropy (CE) criterion:

$$\mathcal{L} = -\sum_t \sum_s l_t(s) \log \mathbf{y}_t(s) \tag{2.8}$$

where $l_t(s)$ is the ground-truth label vector for frame t, and $\mathbf{y}_t(s)$ is the output vector from the softmax layer.

Learning of neural network parameters was performed using back propagation with stochastic gradient descent (SGD) and momentum. SGD is a gradient descent approach with a slight modification where the true gradient is estimated by the gradient of a small subset of the training examples, called a mini-batch.

Another widely used approach for neural network models in ASR is called the tandem approach [39]. It shares the same training paradigm with the hybrid DNN-HMM approach. However, instead of using the target output of DNNs (Eq. 2.7) to replace the likelihoods, the DNN is used to generate input features to feed into another HMM recognizer. The input feature is extracted from the outputs of some hidden layer in the DNN. That layer is forced to have a smaller number of neurons, called a bottleneck (BN) layer. Since the DNN usually learns better representations, BN features usually outperform handcrafted features such as MFCCs and PLPs.

Hybrid vs. Tandem

The tandem system usually requires more training steps and performs worse in CE training due to the bottleneck layer. But the Tandem system still attracts a lot of attention for two reasons: (1) it can build on existing techniques in the HMM world such as speaker adaptation and (2) the BN features can be used in other tasks, e.g. multilingual ASR system for low resource languages.

In practice, our experiments also suggest that a hybrid system is better than a tandem system for a large scale dataset. But for low-resource tasks, a state-of-the-art ASR system needs to train a hybrid system on top of BN features [60].

Therefore, in this thesis, we mostly use the hybrid approach. However, for low-resource language, we extract BN features first and then apply our new neural network models on top of them to reach state-of-the-art performance. In the following subsection, more advanced deep learning models, i.e., stacked BN features, and RNNs, are described for hybrid and tandem ASR systems.

2.3.2 Low-rank Stacked Bottleneck Architecture

The Tandem approach benefits from being able to use existing techniques developed for the GMM-HMM framework such as discriminative training, or speaker adaptation. One can also easily train a tandem system as a feature extractor on a high-resource language, and then use it for a low-resource language. However, as we describe in the previous section, there usually exists a gap in performance between the Tandem approach and the hybrid approach. In [121], we propose a method to improve existing Tandem approaches to be more comparable with the hybrid approach. We will describe this approach in this section, and use it as a baseline system in Chapter 4.

Low-rank matrix factorization

The left side of Figure 2-3 shows a typical ASR DNN architecture. Following Sainath *et al.*, [87] we investigate a low-rank approximation to the weights of the softmax layer of the network. By considering the weights of the softmax layer as a matrix, we

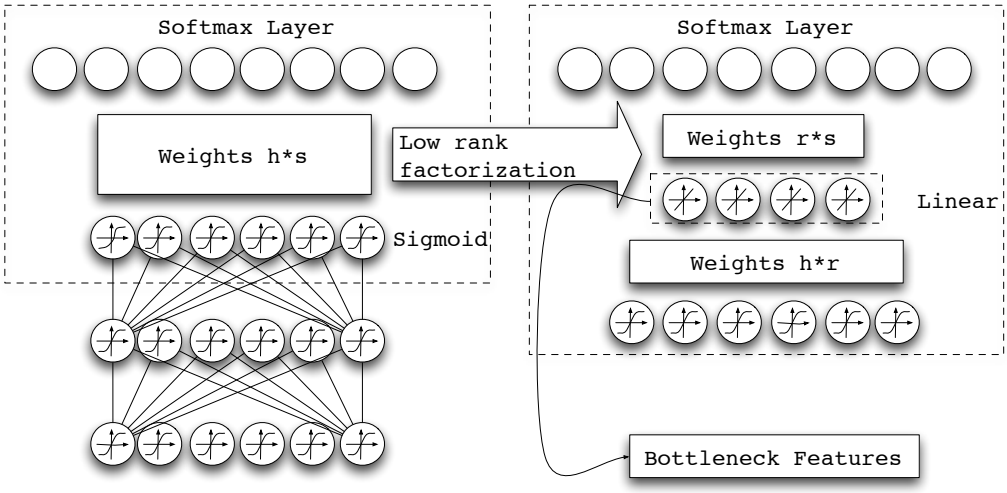


Figure 2-3: Diagram of the low-rank factorized DNN. The left side of the figure represents a DNN with h hidden units and s target labels. The right side of the figure is the low-rank bottleneck DNN. The last hidden layer is now replaced by two set of weights with a linear activation function in between. Bottleneck features can be extracted from the DNN by taking the output of the linear activations.

can factorize the weight matrix into two matrices of lower rank. As illustrated by the right side of Figure 2-3, this is done by replacing the usual softmax layer weights by a linear layer with a small number of hidden units followed by a softmax layer. More specifically, a new BN output layer with r linear hidden units is inserted into the last weight matrix with a hidden layer of size h, and a softmax layer with s state posterior outputs. This changes the number of parameters from $h * s$ to $r * (h + s)$. Notice that there is no non-linearity for this BN output layer. Instead of using this structure for hybrid DNNs, we use it for extracting BN features. There are two benefits for using this method. First, it ensures the best achievable frame accuracy even with a relatively small r. Second, the linearity of the output for the BN layer prevents a loss of information when we treat the DNN as a feature extractor.

Stacked bottleneck (SBN) features

The idea of using hierarchical processing of neural networks (NNs) has been explored by several researchers. Valente *et al.* use a second NN to help correct the posterior

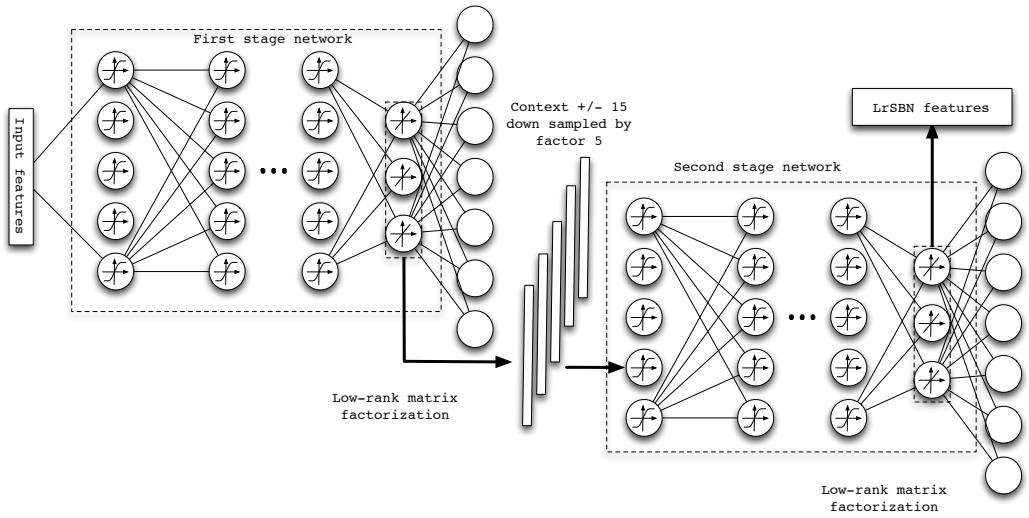


Figure 2-4: Diagram of the stacked bottleneck neural network feature extraction framework. Two DNNs are combined together in a series. Starting from the left side of the figure, original input features are passed to the first low-rank BN network. The activations of the linear layer are extracted and stacked with time offsets: $-10, -5, +5, +10$. The stacked feature is then used as input features to the second BN network. Finally, the LrSBN features can be extracted from the BN layer of the second DNN.

outputs of the first NN by feeding it a different set of features [111]. In the context of the Babel project, SBN features have shown promising results in [59]. One argument for the usage of these cascading structures is that they enable more information, such as additional context, to be utilized more effectively [37].

Low-Rank Stacked Bottleneck (LrSBN)

Figure 2-4 gives an overview of our proposed low-rank SBN feature extraction framework. The BN outputs from the first DNN are concatenated with context expansion and fed to the second DNN. This structure is similar to [59] except that we always place the linear BN layer (for the low-rank factorization) in the *last* hidden layer instead of a sigmoid BN layer in the middle of the network. Experiments in [87] have shown that the hidden layers do not have the same low-rank properties as the weights in the softmax layer. We also use tied-states as the output targets instead of

CI targets.

We use this LrSBN as the baseline feature for our multilingual system described in Chapter 4.

2.3.3 Convolutional Neural Networks

Convolutional neural network (CNNs) have been applied widely in the areas of computer vision [66, 64]. In recent years, CNNs, composed of at least one convolutional layer, have shown improvement over traditional fully-connected deep neural networks on many ASR tasks [88, 3]. Unlike fully connected layers, convolutional layers take into account the input topology, and are designed to reduce translational variance by forcing weight sharing and applying mean/max pooling afterward.

Let input feature $\mathbf{x} \in \mathbb{R}^{T_\mathbf{x} \times F_\mathbf{x}}$ be a two dimensional matrix, where $T_\mathbf{x}$ denotes the context window width and $F_\mathbf{x}$ denotes the number of frequency bands. Suppose there are K kernels with weight $\mathbf{W}_1$, $\mathbf{W}_2$, $\cdots$, $\mathbf{W}_K$ and bias b_1, b_2, $\cdots$, b_K. We use k to index kernels and the k-th kernel $\mathbf{W}_k \in \mathbb{R}^{T_k \times F_k}$. The activation (also called a feature map) of the k-th kernel centered at the (t, f)-position of the input feature is

$$h_{k_{t,f}} = \theta(\sum_{i=1}^{T_k} \sum_{j=1}^{F_k} x_{i+(t-\lceil \frac{T_k}{2} \rceil), j+(t-\lceil \frac{F_k}{2} \rceil)} W_{k_{i,j}} + b_k), \tag{2.9}$$

where θ is the activation function, which we set to be rectified linear units here. Note that we set $x_{i',j'} = 0$ if i', j' exceeds the boundary.

In this thesis, CNNs are investigated and used extensively in Chapter 6.

2.3.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are variants of feed-forward neural networks, which contain feedback loops that feed activations not only to the next layer but also as the input to the current layer at the next time step. This design enables the network to consider all contexts from the past to make a decision about the current frame, which is a desirable property since contextual information plays an important

role in acoustic modeling. However, in early work, the choice of the RNN function is simply a linear transformation followed by element-wise activation. In practice, this simple RNN cannot preserve information over a long period when we use it for acoustic modeling. This is because commonly-used activation functions, such as the sigmoid function and hyperbolic tangent function, compress the input into a small dynamic range; therefore, the activations of the same layer from n-steps before would have been compressed n-times when it arrives at the current time step. Thus the history information has a minor influence. While training with back-propagation through time (BPTT), this problem is also known as the vanishing gradient problem in the sense that error signals are not likely to back-propagate along time dimension for many hops. To address this issue, the LSTM block was proposed in [49] to replace the traditional hidden unit. In this section, we review the basic single-layer long short-term memory (LSTM) RNNs, and their deep versions.

Long Short-Term Memory (LSTM) RNNs

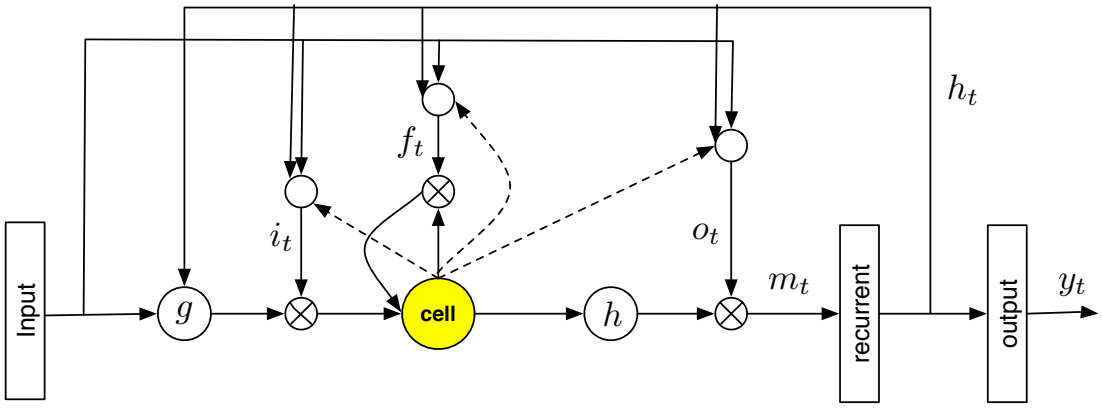


Figure 2-5: Long Short-Term Memory RNNs. An LSTM block contains specialized gates: 1) input gate, i_t, decides whether the input is significant enough to remember, 2) forget gate, f_t, decides whether to continue remembering the value, and 3) output gate, o_t, decides whether it should output the value.

The LSTM RNN was initially proposed in [49] to solve the vanishing gradient problem that often happens when training RNNs. It introduces a linear dependency

between $\mathbf{c}_t$, the memory cell state at time t, and $\mathbf{c}_{t-1}$, the same cell's state at $t-1$. Nonlinear gates are introduced to control the information flow. The operation of the network follows the equations

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{mi}\mathbf{m}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \tag{2.10}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{mf}\mathbf{m}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f) \tag{2.11}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{mc}\mathbf{m}_{t-1} + \mathbf{b}_c) \tag{2.12}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{mo}\mathbf{m}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o) \tag{2.13}$$

$$\mathbf{m}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.14}$$

iteratively from $t = 1$ to $t = T$, where $\sigma(*)$ is the logistic sigmoid function, $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t, \mathbf{c}_t$ and $\mathbf{m}_t$ are vectors that represent values of the input gate, forget gate, output gate, cell activation, and cell output activation at time t, respectively. $\odot$ denotes element-wise product of vectors. $\mathbf{W}_*$ are the weight matrices connecting different gates, and $\mathbf{b}_*$ are the corresponding bias vectors. All these matrices are full except diagonal matrices $\mathbf{W}_{ci}$, $\mathbf{W}_{cf}$, and $\mathbf{W}_{co}$ that connect the cell to gates.

For large-vocabulary continuous speech recognition (LVCSR) tasks, we insert a projection layer, as suggested in [90], to reduce computation. As illustrated in Figure 2-5, the equations for network units change slightly, as the $\mathbf{m}_{t-1}$ activation vector is replaced with $\mathbf{h}_{t-1}$ in Eq. (2.10),(2.11),(2.13), and (2.14):

$$\mathbf{h}_t = \mathbf{W}_{mr}\mathbf{m}_t. \tag{2.15}$$

We then feed this $\mathbf{h}_t$ to the softmax layer, and produce $\mathbf{y}_t$ as the final output:

$$\mathbf{y}_t = \text{Softmax}(\mathbf{h}_t). \tag{2.16}$$

Deep LSTM (DLSTM) RNNs

Deep LSTM (DLSTM) RNNs are formed by stacking multiple layers of LSTM cells. Specifically, the output of the lower layer LSTM cells $\mathbf{h}_t^l$ is fed to the upper input

layer as input $\mathbf{x}_t^{l+1}$. Although each LSTM layer is deep in time since it can be unrolled in time to become a feed-forward neural network in which each layer shares the same weights, deep LSTM RNNs still significantly outperform single-layer LSTM RNNs. It is conjectured [90] that DLSTM RNNs can make better use of parameters by distributing them over space through multiple layers. Note that in conventional DLSTM RNNs the interaction between cells in different layers must go through the output-input connection.

Similar to deep LSTMs, we use deep LSTMs with a projection layer (**LSTMP**), where multiple LSTM layers each with a separate recurrent projection layer are stacked. In this thesis, we will mostly use this system as our baseline.

2.4 Speech Corpora

Eight speech corpora are used in the experiments in this thesis. A brief overview of each speech corpus will be described in the following sections.

2.4.1 TIMIT

The TIMIT corpus is designed to provide speech data for acoustic-phonetic studies and the development and evaluation of ASR systems [27]. The corpus contains broad-band recordings of 438 male speakers and 192 female speakers of eight major dialects of American English, each reading ten phonetically rich sentences. The recorded utterances are stored as 16-bit, 16 kHz speech waveform files, and the time-aligned phonetic transcriptions of the recorded sentences are also provided in the corpus, which is the gold standard we use to evaluate the model.

2.4.2 IARPA-Babel corpus

The IARPA-Babel program focused on ASR and spoken term detection of low-resource languages [1]. The goal of the program was to reduce the amount of time needed to develop ASR and spoken term detection capabilities for a new language.

The data from the Babel program consists of collections of speech from a growing list of languages. For this work, we will consider the full language pack (FLP) (60-80 hours of training data) of the 11 languages released in the first two years as source languages, while the languages in the third year will be the target languages [18]. Some languages also contain a mixture of microphone data recorded at 48kHz in both train and test utterances. For the purpose of this thesis, we downsampled all the wideband data to 8kHz and treated it the same way as the rest of the recordings. For the target languages, we focus on the Very Limited Language Pack (VLLP) condition which includes only 3 hours of transcribed training data. This condition excludes any use of human generated pronunciation dictionary.

2.4.3 AMI

The AMI corpus comprises approximately 100 hours of meeting recordings in instrumented meeting rooms [9]. Multiple microphones were used, including individual headset microphones, lapel microphones, and one or more microphone arrays. In this work, we use the single distant microphone (SDM) condition for our experiments. Our systems follow the split recommended in the corpus release: 80/9/9 hours for train/dev/test respectively. For our training, we use all segments provided by the corpus, including those with overlapping speech. Our models are evaluated on the test set only. NIST's asclite tool [24] is used for scoring.

2.4.4 SWBD

Switchboard-1 (SWBD) corpus [30] was used for our experiments. We used 4,870 sides of conversations (about 300 hours speech) from 520 speakers as training data, and 40 sides of Switchboard-1 conversations (about 2 hours speech) from the 2000 Hub5 evaluation as testing data.

2.4.5 HKUST

The HKUST Mandarin Telephone Speech[2] dataset is a medium-sized corpus containing 150 hours of conversational telephone speech from Mandarin speakers, recorded at an 8kHZ sampling rate. The two callers do not know each other in advance, and topics similar to those in Fisher English were used to initiate a conversation. The release is split into training and development sets with 873 calls and 24 calls respectively, of which we use the development for evaluation.

2.4.6 GALE Mandarin

We merged GALE Phase 2 Chinese Broadcast Conversation Speech , GALE Phase 3 Chinese Broadcast Conversation Speech Part 1 and Part 2, and GALE Phase 2 Chinese Broadcast News Speech[3], to create a large 500-hour Mandarin corpus. All four corpora are recorded at a 16k sampling rate from Chinese broadcast programs. We use the same 3-hour evaluation set as in [52].

2.4.7 Arabic MGB

This dataset is provided by the 2016 Arabic MGB Challenge,[4] containing about 1200 hours of Arabic broadcast programs taken from Aljazeera TV over ten years, recorded at a 16kHZ sampling rate. Ten hours of data are partitioned as the official development set for the challenge and will be used for evaluation in this work.

2.4.8 Wall Street Journal

The Wall Street Journal (WSJ) speech corpus contains read speech of articles drawn from the Wall Street Journal text corpus.[5] An equal number of male and female speakers were chosen to record the corpus for the diversity of voice quality and dialect.

[2]It is available as LDC2005S15 in LDC.

[3]They are available as LDC2013S04, LDC2014S09, LDC2015S06 and LDC2013S08 in LDC respectively.

[4]http://www.mgb-challenge.org/arabic.html

[5]It is available as LDC93S6A and LDC94S13A in LDC.

Two microphones were used for recording: a close-talking Sennheiser HMD414, and a secondary microphone. The speech data were sampled at 16 kHz and saved as sequences of 16-bit data samples. We used the si284 as the training set, dev93 as the validation set and eval92 as the test set.

3

General Purpose Deep Learning Toolkit

3.1 Introduction

Great success has been achieved by moving from simple feed-forward networks to more advanced structures such as CNNs and RNNs [88, 36]. In applications such as speech, where the data is sequential, the most fundamental components are perhaps feature extraction and aggregation. In CNNs, convolutions typically learn feature maps, and in RNNs, recurrent connections from the building block to store or discard features. The success of deep learning (and neural networks) often derives from well-chosen neural components as the building blocks. We need to carefully design the building blocks, such as by using LSTMs to replace simple RNNs, and need to efficiently compose different elements, such as [89], which reduced WER by stacking convolutional and recurrent layers. Conversely, to exploit the structure and information inside a particular task, we need to design *customized* models.

Implementing a neural network from scratch can be nontrivial, and involve a lot of heuristics to avoid pitfalls. For example, just adding one recurrent loop to a feed-forward network requires rewriting all the gradient computations, and can be 20 times slower because of sample-by-sample processing. As a result, an open source, modular, and extensible deep learning system is crucial for the deep learning community – both in industry and academia. It allows people with different expertise to focus on their

own research and leverage advances in other areas.

In this chapter, we describe an open source deep learning toolkit called the computational network toolkit (CNTK) [119]. CNTK is an open source project initiated by Microsoft, jointly developed by many deep learning practitioners and researchers from all over the world. The two main CNTK contributions from the author of this thesis include 1) supporting arbitrary recurrent neural networks, and 2) speeding up training on GPUs. The following sections describe these two contributions in more details.

3.2 Related Work

There are many deep learning toolkits that are comparable in various ways with CNTK [119]. Theano [8], Torch [19], Caffe [56], Chainer [108], TensorFlow [2] and MXNet [15] are a few systems designed primarily for the training of neural networks. When CNTK was first released in 2014, only Theano and Torch had support for recurrent neural networks, but none of them had been optimized for speech recognition, e.g. the large utterance length variation.

The efficient batching algorithm we will propose in Section 3.4.3 relied on the use of a CNTK special feature called a "DelayNode", which we will describe in Section 3.3.2. It enables efficient batching and a dynamic unrolling algorithm for RNNs. However, it is not as flexible as other toolkits to do utterance level operations, e.g. sequence-to-sequence models [103]. The algorithm we propose in Section 3.4.4 and Section 3.4.5 did not depend on any specific toolkit.

3.3 Computational Network

A computational network (CN) [119] can be described as a directed graph where each vertex, called a computation node, represents a computation, and each edge represents the operator-operants relationship. Specifically, leaf nodes in the graph are used to represent input values or model parameters, and non-leaf nodes represent

some computational relations to its children.

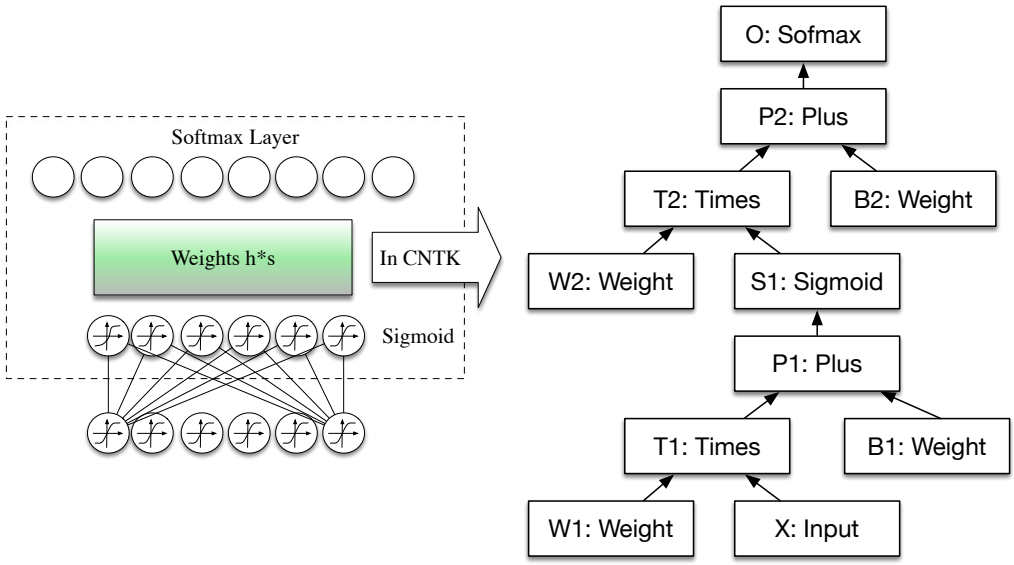


Figure 3-1: A single-hidden-layer neural network and its computational network representation. The left side of the picture represents a typical single layer DNN with h hidden units and s target labels. The right side of the picture shows the its CN representations. All the weight matrices map to leaf nodes and all the operations map to non-leaf nodes.

All of the popular NNs such as DNNs, CNNs, and RNNs can be reduced to a series of computational steps. If we know how to compute each step as well as the order in which they are computed, we have an implementation of these models. This observation suggests that we can generalize and treat all these models as special cases of CNs. For example, Figure 3-1 is the CN representation of a one-hidden-layer sigmoid neural network. In the right side of the graph, each node n is identified as a pair $name : operator$. We can see a non-leaf node defines a computation such as $P1$, which represents "Plus", and the leaf node $B1$, which represents a weight matrix.

Note that CNs can cover a significantly larger variety of, and more complicated models than standard neural models such as DNNs and RNNs. To support this architectural flexibility, we need to employ special algorithms to evaluate and train CNs.

3.3.1 Forward Computation

Algorithm 1 Forward computation of CN without loop

```
1: procedure ForwardOrderWithOutLoop(root)
2:     G ← Computation Graph
3:     V ← All the computation nodes
4:     visited ← {}
5:     order ← {}
6:     if root ∉ visited then
7:         visited ← visited ∪ root
8:         for all v ∈ root.children do
9:             ForwardOrderWithOutLoop(v)
10:        order ← order + root
```

Given the model parameters (i.e., weight nodes in Figure 3-1) and input values, we can compute the value of any node. In this DNN case, the computational order can be trivially determined by layer-by-layer computation from the bottom up. In a more general case, when a CN is a directed acyclic graph (DAG), the computation order can be determined with a depth-first traverse over the DAG, as illustrated in Algorithm 1. Here, the computation of the next node starts only after the computation of the previous node has finished. But we can easily extend this framework in the asynchronized case, as long as there is no dependency between two nodes.[1] Once the order is decided, it will remain the same for all subsequent runs, regardless of the computational environment. In other words, this algorithm only needs to be executed per output node and then the computational order can be cached.[2]

In this case, each computation node only needs to know how to compute its value when the operands are known. The computation can be as simple as matrix summation, or element-wise application of the sigmoid function, or as complex as a multiple layer NN, as long as the CN is still a DAG. The conditions change when we introduce a recurrent loop in the graph. In Section 3.3.2, we will describe a simple

[1]For more details about forward computation in CNs over DAG, please check the CNTK book [119].

[2]Recently, dynamic architectures (e.g., where every training instance has a different model architecture) have become more and more popular for natural language processing tasks. For the construction of dynamic graphs, readers are invited to refer to PyTorch (`http://pytorch.org/about/`) for more details.

version of the forward computation algorithm for a CN with a recurrent loop. In Section 3.4, we will describe several speed up approaches for RNNs.

3.3.2 Recurrent Connections

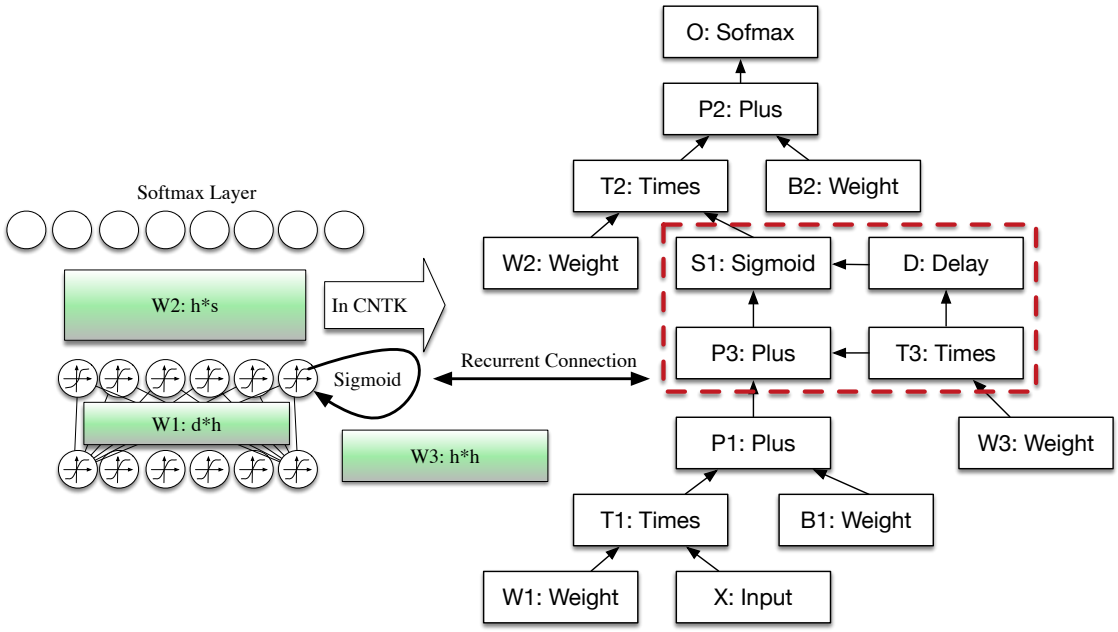


Figure 3-2: A single-hidden-layer neural network with a recurrent loop and its computational network representation. The left side of the picture represents a typical single layer RNN with one recurrent loop from the hidden layer to itself. The right side of the picture shows the its CN representations. The recurrent loop is in the red dashed-line box and the weight matrix W3 of this recurrent loop is mapped to a leaf node of this red box.

In CNs, the recurrent connection can be specified using a *Delay* node that retrieves the value d samples in the past, where each column of $\mathbf{x}$ is a separate sample stored in the descending order of time: $\mathbf{h}_t(d, \mathbf{x}) = \mathbf{x}_{(t-d)}$.

Figure 3-2 illustrates the correspondence between the simple RNN and the CN representation using a *Delay* node. The operations performed by the neural network

at time t are:

$$\mathbf{P}_t = \mathbf{W}_1\mathbf{x}_t + \mathbf{B}_1 \tag{3.1}$$

$$\mathbf{S}_t = \sigma(\mathbf{W}_3 s_{t-1} + \mathbf{P}_t) \tag{3.2}$$

$$\mathbf{O}_t = \mathbf{W}_2\mathbf{S}_t + \mathbf{B}_2 \tag{3.3}$$

The recurrent loop is depicted in the dashed-line box in Figure 3-2 and relies on a *Delay* node to indicate the dependence of the hidden layer activity on its past values.

In the DAG case, node evaluation under this condition is very efficient because many samples can be computed concurrently. However, a CN with a loop cannot be computed for a sequence of samples as a batch since the next sample's value depends on the previous samples. A simple way to do the forward computation and backpropagation in a recurrent network is to unroll all samples in the sequence over time. Once unrolled, the graph can be expanded into a DAG, and the forward computation as in Algorithm 1 can be directly applied. This means, however, that all computation nodes in the CN need to be computed sample-by-sample, and this significantly reduces the potential for parallelization.

In the next section, we will discuss how can we speedup the computation of a CN with directed loops.

3.4 Efficient Network Training

Nowadays, GPUs are widely used in deep learning by leveraging massive parallel computations via mini-batch based training. For unidirectional RNN models, to better utilize the parallelization power of the GPU card, in [119], multiple sequences (e.g., 40) are often packed into the same mini-batch. Truncated BPTT is usually performed for parameter updating, therefore, only a small segment (e.g., 20 frames) of each sequence has to be packed into the mini-batch. However, when applied to sequence level training (Bidirectional LSTM (BLSTM) or sequence training), a GPU's limited memory restricts the number of sequences that can be packed into a mini-

batch, especially for LVCSR tasks with long training sequences and large model sizes. An alternative way to speed up processing is by using asynchronous SGD based on a GPU/CPU farm [45]. In this section, we are more focused on fully utilizing the parallelization power of a single GPU Card. The algorithms proposed here can also be applied to a multi-GPU setup.

3.4.1 Sample-by-Sample Processing Only Within Loops

Algorithm 2 Forward computation of an arbitrary CN

1: **procedure** FORWARDORDERWITHRECURRENTLOOP($root$)
2: $\mathcal{G} \leftarrow$ Computation Graph
3: $\mathcal{V} \leftarrow$ All the computation nodes
4: $StronglyConnectedComponentsDetection(G, G')$ [107] ▷ G' is a DAG of SCC
5: $ForwardOrderWithOutLoop$(G') $\rightarrow order$
6: **for** *all* SCC $\in G', v \in$ SCC **do**
7: $v.order = \max\{v.order \in \text{SCC}\}$
8: **for** *all* $s.root \in G' \wedge s.isloop()$ **do**
9: $GetLoopForwardOrder(s) \rightarrow order$ for each SCC
return *order* for DAG and *order* for each SCC
10: **procedure** GETLOOPFORWARDORDER(ROOT)
11: Set delayNode as leaf ▷ Convert the loop to DAG
12: $ForwardOrderWithOutLoop(root)$

To speed up training, we can identify loops in the CN and only apply sample-by-sample computation for nodes inside the loops. For the rest of the computation nodes, all samples in the sequence can be computed in parallel as a single matrix operation. For example, in the red dashed-box of Figure 3-2, all the nodes included in the loop of $T_{t-1} \rightarrow P3 \rightarrow S1 \rightarrow D \rightarrow T_t$ need to be computed sample by sample. All the rest of the nodes can be computed in batches. In other words, we need to identify the cliques where all the nodes depend on others (forming a loop). This procedure is equivalent to finding all the strongly connected components (SCC) in the graph. A graph is said to be strongly connected if every vertex is reachable from every other vertex. A SCC of a directed graph is a maximal strongly connected subgraph. We

can use Tarjan's algorithm to easily find all the SCCs in a directed graph.[3] Once the loops are identified, they can be treated as composite nodes in the CN, and the CN is reduced to a DAG. All the nodes inside each loop (or composite node) can be unrolled over time and also reduced to a DAG. For all these DAGs, the forward computation and backpropagation algorithms we discussed in the previous section can be applied. The detailed procedure for determining the forward computation order in the CN with arbitrary recurrent connections is described in Algorithm 2. Since the input to the *Delay* nodes is computed in the past, they can be considered as leaf nodes if we only consider one time slice, which makes the order decision inside loops much easier.

3.4.2 Processing Multiple Utterances Simultaneously

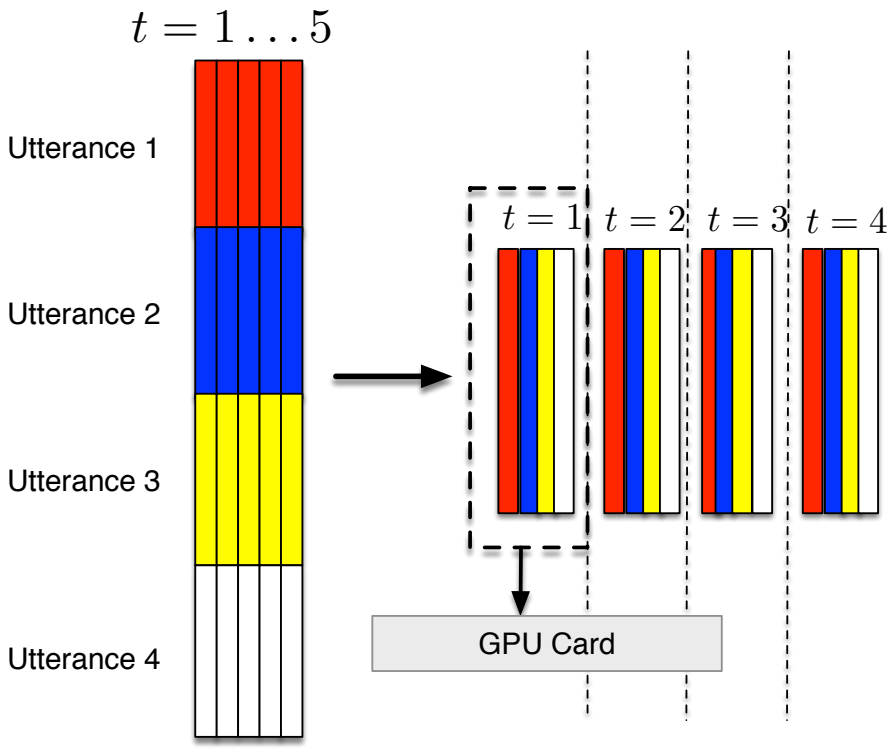


Figure 3-3: Processing multiple sequences in a batch (an example with 4 sequences). Each color represents one sequence. Frames on the same time step from different sequences are grouped together and computed in batch.

[3]Tarjan's strongly connected components algorithm has a complexity of $\mathcal{O}(|V| + |E|)$ and can easily detect all the SCC in a graph.

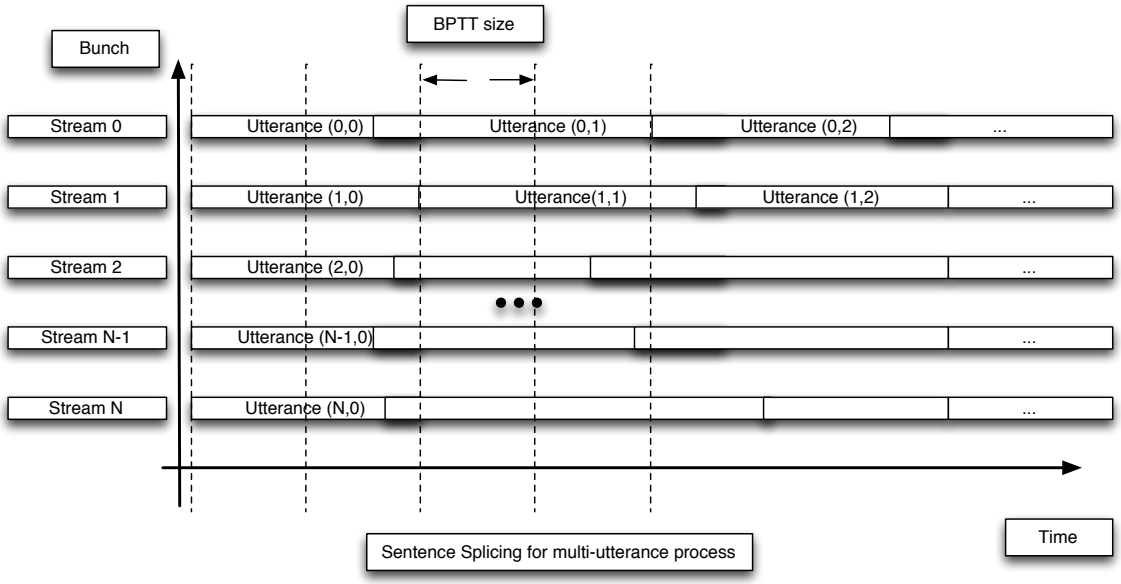


Figure 3-4: Truncated BPTT training with sentence splicing. When we reach the end of a sentence for each stream, we will load a new sentence, instead of appending garbage frames.

We can further speed up the training within the loop by processing multiple sequences at a time. To implement this, we need to organize sequences so that the frames with the same frame id from different sequences are grouped together, as shown in Figure 3-3. By organizing sequences in this way, we can compute frames from different sequences in a batch when inside a loop, and compute all samples in one batch outside loops. For example, as illustrated in Figure 3-3, we can compute 4 frames together for each time step. Note that sentence length variation in the training data requires that the number of samples in the input matrix to be the maximum sentence length in the training corpus. To handle this issue, "garbage frames" are appended to the end of shorter sentences in the batch. These redundant frames are ignored during BPTT.

3.4.3 RNN Training with Sentence Splicing

As described in the previous section, as the ratio between the maximum and average sentence length of the training data increases, the potential speed up from paralleli-

sation decreases, since there is redundant computation for the "garbage" frames. To improve efficiency, we load a new utterance and keep track of the cell state during forward computation. Instead of appending garbage frames at the end of each sentence, each stream now contains a sequence of concatenated sentences, as illustrated in Figure 3-4. In this case, we only have redundant computation at the end of each epoch. The additional cost is keeping track of the sentence boundary for each utterance. This can be very inefficient if we have to check this for every node. However, only the *Delay* nodes need to access this information and reset the state when they are aware of a sentence boundary in the current slice. Our experiments show that in a standard 3-layer LSTM with 1,000 nodes, the cost of the reset boundary in the *Delay* node is less than 0.1%. In comparison, "garbage" frames can amount to more than 40% in a speech corpus due to sentence length variation.

3.4.4 Latency-controlled bi-directional model training

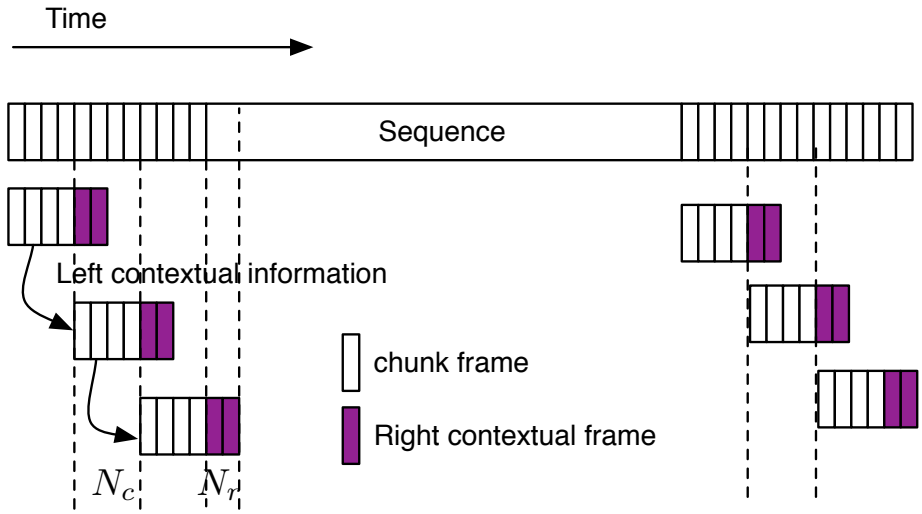


Figure 3-5: Truncated BPTT for Latency-controlled bi-directional RNN. We keep track the left contextual information in the forward RNN for the entire sequence when we move to the next BPTT chunk. We use a latency parameter N_r to control the length of right contextual information in the backward RNN.

The batching algorithm we described in the previous section works only for the unidirectional case. Training bi-directional RNNs, or sequence-discriminative training

of memory-demanding neural networks, often requires whole sequence-based BPTT, which is problematic is because the number of sequences that can be packed into the same mini-batch is usually quite restricted due to GPU memory limit. This situation can significantly decrease the training and evaluation speed. To speed up the training of bi-directional RNNs, the Context-sensitive-chunk BPTT (CSC-BPTT) was proposed in [14]. In this method, a sequence is first split into chunks of fixed length N_c. Then N_l past frames and N_r future frames are concatenated before and after each chunk as the left, and right context, respectively. The appended frames are only used to provide contextual information, and do not generate error signals during training. Since each chunk can be independently drawn and trained, they can be stacked to form large minibatches to speed up training.

Unfortunately, the model trained with CSC-BPTT is no longer a true bidirectional RNN, since the history it can exploit is limited by the left and right context that is concatenated to the chunk. It also introduces additional computational cost during decoding, since both the left and right contexts need to be recomputed for each chunk.

To solve these problems in CSC-BPTT, we propose latency-controlled bi-directional RNNs. Unlike CSC-BPTT, in our new model we incorporate the entire history while still using a truncated future context as illustrated in Figure 3-5. Instead of concatenating and computing N_l left contextual frames for each chunk, we directly carry over the left contextual information from the previous chunk of the same utterance. For every chunk, both the training and decoding computational cost is reduced by a factor of $\frac{N_l}{N_l+N_c+N_r}$. Moreover, loading the history from the previous mini-batch instead of using a fixed contextual window makes the context exact when compared to the uni-directional model. Note that standard BLSTM RNNs come with significant latency, since the model can only be evaluated after seeing the whole utterance. In the latency-controlled BLSTM (LC-BLSTM) RNNs, the latency is limited to N_r, which can be set by the users. In our experiments, we process 40 utterances in parallel, which is 10 times faster than processing the whole utterance, without performance loss. Compared to the CSC BPTT, our approach is 1.5 times faster and often leads to better accuracy.

3.4.5 Two-forward-pass Method for Sequence Training

Algorithm 3 TwoForwardPassSequenceTraining

```
1: procedure TwoForwardPassSequenceTraining()
2:     S ← Sequencess
3:     A ← Alignments corresponding to S
4:     D ← Denominator lattices corresponding to U
5:     M ← MinibatchReader(S)          ▷ (E.g., 40 sequences, each with 20 frames)
6:     P ← SequencePool(S, A, D)
7:     for all m ∈ M do
8:         if P.HasGradient(m) then
9:             g ← P.Gradient(m)
10:            forward_pass(m)
11:            set_output_node_gradient(g)
12:            backward_pass(m)
13:            parameter_update()
14:        else
15:            m_p ← M.CurrentMinibatchPointer()
16:            while P.NeedMoreMinibatch() do
17:                m_1 ← m_p.ReadMinibatch()
18:                p ← forward_pass(m_1)          ▷ Posterior from forward-pass
19:                P.ComputeGradient(m_1, p)
20:            M.ResetMinibatchPointer(m_p)
```

For sequence training, however, we cannot apply the truncated training version to the sequence-discriminative training of RNNs and other memory-hungry models such as deep convolutional neural networks (CNNs), since the signal computation itself requires having posteriors for the whole sequence. One way to speed up the training, in this case, is to use asynchronous SGD on a GPU/CPU farm [45], at the cost of low computing resource utilization on each GPU/CPU. This solution is of course not ideal, and GPU/CPU farms can be quite expensive to build and maintain.

In this section, we propose a two-forward-pass method for efficient sequence-discriminative training of memory-hungry models. Algorithm 3 demonstrates the pseudocode of the proposed method. The general idea is to enable partial sequences in each minibatch for sequence-discriminative training. For this to happen, we will have to maintain a sequence pool, and compute the gradient for those sequences from their corresponding lattices in advance. This requires an additional forward-pass so

that gradients from lattices can be computed at the sequence level, and stored in the sequence pool (thus the name two-forward-pass). After preparing the gradients in the sequence pool, sequences can again be split into small segments (e.g., 20 frames), and segments from multiple sequences (e.g., 40) can be packed into the same mini-batch for efficient parameter updating.

3.5 Experiments

We use the AMI [9] and SWBD [30] corpus for all our experiments. Kaldi [79] and CNTK [119] are used for ASR system building. Details of the corpora are in Chapter 2 and system descriptions are similar to the experimental section in Chapter 5.

3.5.1 Speed

As we mentioned above, the motivation of the two-forward-pass method is to allow more utterance parallelization in each mini-batch when performing sequence-discriminative training for recurrent neural networks. On the NVIDIA Grid K520 GPUs that we experiment with, we were only able to parallelize at most 4 utterances in the same mini-batch for our given LSTMP network structure. In comparison, we can parallel more than 160 utterances using 20 as BPTT size.

Table 3.1: Speed performance (hours per epoch) comparison of LSTMP sMBR training with and without parallelization in each mini-batch. Experiment is conducted on a $10K$ utterance subset of the AMI SDM task, with NVIDIA Grid K520 GPUs.

	#utterances in each mini-batch	
System	1	40
LSTMP sMBR	13.7	0.75

Table 3.1 compares the training time of the conventional whole utterance approach without multi-utterance parallelization with our proposed two-forward-pass method with 40 utterances processed in the same mini-batch. Since the training of the conventional whole utterance approach is quite time consuming, we conducted

the comparison on a $10K$ utterance subset of the SDM task. From the table, we can see that we get an 18x speedup by using our proposed two-forward-pass method. Further speed improvement is possible by increasing the number of utterances processed in the same mini-batch. For example, in our experiments, we have processed 80 utterances in the same mini-batch without hurting performance. We also found this can be combined with the LC-BLSTM approach, as shown in the experiments in Chapter 5.

3.5.2 Performance

Latency-controlled BLSTMP

Table 3.2: Performance (% WER) comparison of BLSTMP and LC-BLSTMP. The term "(30+30)" means $N_c = 30$ and $N_r = 30$.

Model	WER	Speed
LSTMP	14.0%	1x
BLSTMP	13.0%	18.5x
LC-BLSTMP (30+30)	13.3%	2x
LC-BLSTMP (30+15)	13.6%	1.8x

Table 3.2 reports the WER[4] of LC-BLSTMP models compared to BLSTMP on the SWB corpus. It is clear that the LC-BLSTMP model can reduce the latency without significant performance degradation. It also gives 10 to 15x speed up based on how many utterances are running in parallel.[5]

Two-forward-pass Method for Sequence Training

Table 3.3 reports the WER of sMBR models trained with the proposed two-forward-pass method. The "LSTMP" in this table refers to the LSTM model with a projection layer, while "BLSTMP" is its bi-directional version.

[4]The number was worse than the results we report in Chapter 5 because here we use a different HMM baseline.

[5]The reported number was on K20 cards. We can run more utterances in parallel for the BLSTMP case on more advanced GPUs.

Table 3.3: Performance (% WER) comparison of LSTM sMBR models trained on AMI SDM task using the two-forward-pass method.

System	WER
LSTMP CE	50.7
LSTMP sMBR	49.3
LC-BLSTMP CE	47.3*
LC-BLSTMP sMBR	45.6*

* Experiments were conducted after the JSALT15 workshop with the latest CNTK which may give slightly better results than what we obtained at the workshop.

It is clear from the table that the sMBR sequence-discriminative training criterion consistently improves upon the cross-entropy model. Here we didn't compare with the single-pass method because it takes too long to train the model.

3.6 Summary

In this Chapter, we have demonstrated the efforts we made for a new deep learning toolkit "CNTK" which enables creating arbitrary neural networks easily. We then proposed several algorithms to speed up RNN training based on this framework. More specifically, we first presented an efficient batching algorithm for unidirectional RNN training. Second, we presented the latency-controlled BLSTM to speed up training, and reduce latency for bidirectional LSTMs. We further proposed a two-forward-pass method for sequence-discriminative training of memory-hungry neural networks, which enables more utterance parallelization in each mini-batch, and dramatically decreases the training time. We demonstrated the effectiveness of these method on the AMI SDM task with various recurrent neural network architectures.

CNTK allows us to define complex CNs and to train and evaluate the model. It can significantly reduce the effort needed to develop new models and therefore speed up the innovation. In the following chapters, we will use CNTK to develop new models for speech recognition.

4

Building Feedback Mechanism for Low-Resource Language Speech Recognition

4.1 Introduction

With the emerging trends of smart devices and big data, ASR research has grown significantly over the past decade. However, there are only around 100 languages with speech recognition capability amongst the 7000 languages spoken around the world [82], even though companies and government agencies have invested heavily in speech technology. One reason for this discrepancy is that creating a reliable recognition system requires a large amount of annotated data and linguistic knowledge. The standard recipe for building a speech recognizer typically requires thousands of hours of transcribed speech for training the acoustic model, which could easily take a year just to collect and annotate the data. Therefore, data scarcity remains a big challenge for low-resource language speech recognition.

One way to overcome this issue is to identify and utilize the acoustics-phonetic similarity between languages in order to introduce data sharing from rich resource languages in a multilingual manner to alleviate the data requirements [17]. Instead of leveraging additional language resources, we propose to add more contextual in-

formation when we have limited resources. For example, a speaker adapted feature could give 6% relative gain in WER in *AMI* (≈100hrs) using an LSTM model [106]. However, almost no gain is observed in Table 5.7 when we apply speaker adaptation to *SWBD* (≈300hrs). We believe this is because, in the limited data condition, the model lacks the ability to learn contextual information. Therefore, we propose to improve DNN models for this challenging condition by modifying the neural network structure to incorporate more predicted contextual information.

The behavior of prediction, adaptation, and correction is widely observed in human speech recognition [100, 99]. For example, listeners may guess what you will say next and wait to confirm their guess. They may adjust their listening effort by predicting the speaking rate and noise condition based on current information, or predict and adjust a letter to sound mapping based on the talker's pronunciations. In [100], the research in neural science community shows that a striking feature of human perception is that our subjective experience depends not only on sensory information from the environment but also on our expectations. Inspired by this, we build a neural network model to emulate these prediction and correction feedback mechanisms. We believe by incorporating this prediction information, the network could be more robust on the limited data condition as suggested in [99].

4.2 Related Work

The core concept of prediction, adaptation, and correction has been widely used in models such as the Kalman filter [58]. Through multi-pass decoding, traditional ASR systems also implicitly exploit prediction information.

The work that is most similar to ours is that presented in [55]. Their model explicitly predicts multi-frame state labels that are used during decoding by means of an autoregressive product. Their work, however, does not involve recurrent feedback, and thus cannot effectively utilize long-range dependencies in the signal. More importantly, their model is very different from ours in spirit, and was proposed from a completely different angle.

RNNs can be naturally used to perform prediction. However, both the simple RNN [114, 23], and the LSTM [90, 36] RNN that have been successfully applied to AMs do not explicitly make future predictions. In this study, we have included results from using the simple RNN and LSTM and showed that the PAC-RNN could do better by modeling prediction information more explicitly.

4.3 Prediction-Adaptation-Correction RNNs

In this section, we describe the prediction-adaptation-correction RNN (PAC-RNN), in which a primary (or correction) DNN estimates the state posterior probability based on both the current frame information, and the prediction made from past frames by a prediction DNN. The result from the primary DNN is fed back into the prediction DNN to make better predictions for future frames. In this model, we can consider that, given new, current frame information, the primary DNN makes a correction on the prediction made by the prediction DNN. Alternatively, it can be considered that the primary DNN's behavior is adapted based on the prediction made by the prediction DNN. Although the concept of prediction-adaptation-correction is not new and arises naturally from Kalman filters [58] for example, our specific architecture and its application to ASR are novel.

4.3.1 Model Structure

Figure 4-1 illustrates the structure of the PAC-RNN studied in this thesis. At the center of the model is a main (or correction) DNN and a prediction DNN. The main DNN estimates the state posterior probability $p^{corr}(s_t|\mathbf{o}_t, \mathbf{x}_t)$ given $\mathbf{o}_t$, the observation feature vector, and $\mathbf{x}_t$, the information from the prediction DNN, at time t. The prediction DNN predicts some future target information. In this study, it predicts the posterior probability $p^{pred}(lz_{t+n}|\mathbf{o}_t, \mathbf{y}_t)$ given $\mathbf{o}_t$ and $\mathbf{y}_t$, the information from the correction DNN, where l can be a state s or a phone θ, and n is the number of frames in the look ahead. Note that since $\mathbf{y}_t$ contains information from the correction DNN, and depends on $\mathbf{x}_t$, information from the prediction DNN, and vice versa, a recurrent

loop is formed.

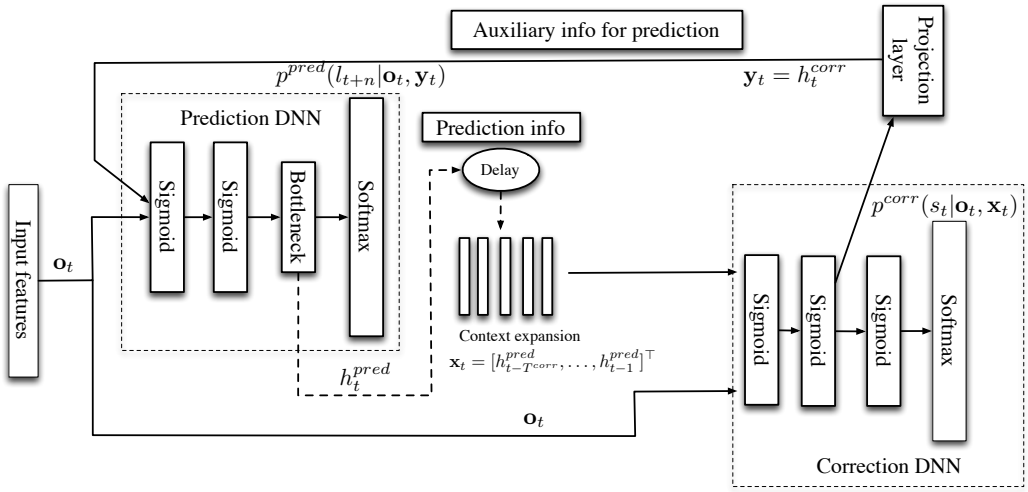


Figure 4-1: The Structure of the PAC-RNN. It includes a prediction model, a correction model, and recurrent loops to link them together.

Information from the prediction and correction DNNs can be drawn from either the softmax layer or a hidden layer. In large vocabulary speech recognition (LVSR) tasks there are often over 5000 states. In these cases, drawing information from the softmax layer can significantly increase the model size. For this reason, we obtained information from a (bottleneck) hidden layer whose size can be set independent of the state size so that the same architecture can be applied to LVCSR tasks directly.

In a very basic setup, $\mathbf{x}_t$, the information from the prediction DNN, is simply the bottleneck hidden layer output value h_{t-1}^{pred}. To exploit additional predictions made in the past, however, we can stack multiple hidden layer values as

$$\mathbf{x}_t = [h_{t-T^{corr}}^{pred}, \ldots, h_{t-1}^{pred}]^T, \tag{4.1}$$

where T^{corr} is the contextual window size used by the correction DNN and is set to 10 in our study. Similarly, we can stack multiple frames to form $\mathbf{y}_t$, the information from the correction DNN, as

$$\mathbf{y}_t = [h_{t-T^{pred}-1}^{corr}, ..., h_t^{corr}]^T, \tag{4.2}$$

where T^{pred} is the contextual window size used by the prediction DNN and is set to 1 in our study. In addition, in the specific example shown in Figure 4-1, the hidden layer output h_t^{corr} is projected to a lower dimension before it is fed into the prediction DNN.

4.3.2 PAC-RNN-LSTM

LSTMs have improved speech recognition accuracy on many tasks over DNNs [36, 90]. To further enhance the PAC-RNN model, we use an LSTM to replace the DNN used in the correction model. The input of this LSTM is the acoustic feature $\mathbf{o}_t$ concatenated with the information from prediction model, $\mathbf{x}_t$:

$$p^{corr}(s_t|\mathbf{o}_t, \mathbf{x}_t) = \text{LSTM}_{corr}([\mathbf{o}_t, \mathbf{x}_t]^\top) \tag{4.3}$$

The prediction model can also be an LSTM but we did not observe performance gain on the experiments. To keep it simple, we use the same DNN prediction model.

4.3.3 Training and Decoding

To train the PAC-RNN, we need to provide supervision information to both the prediction and correction DNNs. As we have mentioned, the correction DNN estimates the state posterior probability, and thus the state label and the frame cross-entropy (CE) criterion can be used. For the prediction DNN, however, we have the freedom to choose either the state or the phoneme label. We will compare the performance difference between these two choices in Section 4.4.

The PAC-RNN training problem is a multi-task learning problem. The two training objectives can be combined into a single one as

$$J = \sum_{t=1}^{T} (\alpha * \ln p^{corr}(s_t|\mathbf{o}_t, \mathbf{x}_t) + (1-\alpha) * \ln p^{pred}(l_{t+n}|\mathbf{o}_t, \mathbf{y}_t)), \tag{4.4}$$

where α is the interpolation weight and is set to 0.8 in our study unless otherwise stated, and T is the total number of frames in the training utterance.

During the decoding stage, the state posteriors (or the scaled likelihood scores converted from them) from the correction DNN are treated as the emission probability similar to that in a typical DNN/RNN-HMM hybrid system [20].

4.4 Experiments on TIMIT

In this section, we evaluate the PAC-RNN on the *TIMIT* phone recognition task. In our experiments the training labels are obtained through forced alignment using our GMM-HMM system trained with the maximum-likelihood criterion.[1] The standard 462-speaker training set is used, and all SA sentences are removed in order to conform to the standard setup as in [74]. A separate development set of 50 speakers is used for tuning all hyper parameters. Results are reported on the 24-speaker core test set, which has no overlap with the development set.

4.4.1 Results Summary

Table 4.1 summarizes the phone accuracy achieved with different hybrid models evaluated in this study. All the DNN/RNN models use a 123 dimensional acoustic feature vector, consisting of 40 dimensional mel-frequency log-filterbank features (FBANK), an energy measure, and their first and second temporal derivatives. In our experiments, 183 target class labels, corresponding to three states for each of 61 phones, are used. A bi-gram phone language model estimated from the training set is used in decoding. The language model weight is tuned on the development set.

We consider three baseline hybrid systems: a DNN with two 2048-unit hidden layers, a simple RNN with two 2048-unit hidden layers in which the final hidden layer is a recurrent layer, and an LSTM with 1024 memory cells. We don't see further performance improvements by increasing the model sizes of these baseline systems.

In the PAC-RNN (S) model, the prediction DNN has a 1024-unit hidden layer and an 80-unit bottleneck layer. The correction DNN has two 1024-unit hidden layers.

[1] In [74], the expertly-annotated phone boundaries were used to generate the training labels, which outperform the HMM generated labels. We use the HMM generated labels since this is the only label available in other datasets.

The projection layer from the correction DNN's hidden layer contains 500 units. In the PAC-RNN (L) model, all 1024-unit hidden layers are replaced with 2048-unit hidden layers.

For the DNN, the simple RNN, and the PAC-RNN models, the input contains a 7-1-7-frame contextual window, which translates to a total size of $123 * 15 = 1845$ dimensions. No context expansion is used for the LSTM model since the best performance is obtained without any context expansion.

All models are randomly initialized without either generative or discriminative pretraining [92]. No momentum is used for the first epoch, and a momentum of 0.9 is used for all subsequent epochs. We have found that turning off momentum for the first epoch helps improve the performance of the final model, although the model performance seems to be worse after the first epoch. We believe this is because the randomly initialized model is highly non-optima, so a noisier gradient helps to move the model to a better starting point. To train the DNN, a learning rate of 0.1 per minibatch is used for the first epoch. The learning rate is increased to 1.0 at the second epoch, after which it is kept the same until the development set training criterion no longer improves, under which condition the learning rate is halved. A similar schedule is used to train the RNNs, except that all the learning rates are reduced to 1/10 of that used for DNN training. Following [74], the state posteriors are directly used as emission probabilities in the HMM without being converted to scaled likelihoods, although conversion on scaled likelihood is preferred for LVSR tasks.

As shown in Table 4.1, the simple RNN only slightly outperforms the DNN, which is consistent with other reported results [114]. The LSTM further improves upon the simple RNN. The PAC-RNN (L) outperforms DNN, RNN and LSTM with 2.4%, 2.1%, and 1.9% absolute phone accuracy improvement, respectively, on the core test set.

4.4.2 Effect of Expanding Prediction Information

As described in Section 4.3, the prediction information fed into the correction DNN can include multiple past predictions. Table 4.2 compares phone error rate with

Table 4.1: TIMIT Phone Error Rate achieved with Different Hybrid Models. PAC-RNN (S) has 1,024 units for each layer and PAC-RNN (L) has 2,048 units for each layer.

Model	Dev	Test	# of Parameters
DNN	20.4%	22.2%	8.4M
Simple RNN	20.5%	21.9%	12.5M
LSTM	20.1%	21.7%	5.9M
PAC-RNN (S)	18.9%	20.0%	6.9M
PAC-RNN (L)	18.4 %	19.8%	15.1M

and without prediction information expansion. From the table, we can observe that if the prediction DNN only predicts the state of the next frame, no gain over the baseline DNN is observed. This is because most frames have the same label as the next frame, and so the prediction DNN does not provide much information to the correction DNN. If the prediction DNN predicts the state of the $t + 10$-th frame, however, we can observe a 1.0% phone accuracy improvement over the DNN baseline. An additional improvement of 0.7% is obtained if 10 past predictions are used by the correction DNN. This indicates that the contextual expansion of the prediction information can be very helpful.

Table 4.2: TIMIT Phone Error Rate achieved with different prediction information expansion. Target such as s_{t+n}, means predicts the state of the $t + 10$-th frame.

Model	Target	Context Expansion	Dev	Test
DNN	-	-	20.4%	22.2%
PAC-RNN (S)	s_{t+1}	no	20.3%	22.3%
PAC-RNN (S)	s_{t+10}	no	19.7%	21.2%
PAC-RNN (S)	s_{t+10}	yes	19.2%	20.5%

4.4.3 Effect of Different Prediction Targets

In order to determine the best prediction target we compared the PAC-RNN with different prediction targets, all with 10-frame contextual expansion, in Table 4.3. From the table we can see that predicting a longer future (e.g., next phone) is better

than predicting a shorter future (e.g., next state), and predicting a more meaningful unit (e.g., using the next phone as the target) is better than predicting the states over a fixed window size (e.g., the state of the $t + 10$-th frame). The best result is obtained by predicting the next phone. The PAC-RNN (S) outperforms the DNN with 2.2% accuracy improvement while PAC-RNN (L) introduces an additional 0.2% improvement on the core test set.

Table 4.3: TIMIT Phone Error Rate achieved using different prediction targets. All the systems have 10 frames of expansion for prediction information.

Model	Target	Dev	Test
DNN	-	20.4%	22.2%
PAC-RNN (S)	Next state symbol	19.8%	21.4%
PAC-RNN (S)	State of the $t + 10$-th frame	19.2%	20.5%
PAC-RNN (S)	Next phoneme symbol	18.9%	20.0%
PAC-RNN (L)	Next phoneme symbol	18.4%	19.8%

4.4.4 Effect of the Recurrent Loop

In this subsection we investigate the effect of the recurrent loop in the PAC-RNN. In Table 4.4, the setup with the recurrent loop is the PAC-RNN we have described in Section 4.3. In the setup with no recurrent loop, the connection from the correction DNN back to the prediction DNN is removed, while the prediction DNN is still used to provide prediction information to the correction DNN. From the table, we can observe that including the recurrent loop is critical, and can achieve 1.8% accuracy improvement over the system with no recurrent loop, which is only 0.4% better than a DNN. We believe this is because long-range information can be exploited more effectively with recurrent connections.

4.4.5 Effect of Optimizing the Prediction Criterion

To train the PAC-RNN we optimize a combined objective function that is an interpolation of the correction and prediction criteria as shown in Eq. 4.4. By adjusting

Table 4.4: TIMIT Phone Error Rate achieved with or without recurrent loop.

Model	With Recurrent Loop	Dev	Test
DNN		20.4%	22.2%
PAC-DNN (S)	No	20.0%	21.8%
PAC-RNN (S)	Yes	18.9%	20.0%

the interpolation weight α we can change the relative importance of each criterion. Table 4.5 summarizes the phone recognition accuracy achieved when the PAC-RNN is trained with different interpolation weights. As expected, if we set α to 1.0 to remove the prediction criterion from the training objective function, the PAC-RNN performs almost as well as the simple RNN and LSTM. If α is set to a small value (e.g., 0.6 in the table) such that the main criterion is not sufficiently emphasized, the performance also degrades, but it remains better than the PAC-RNN trained without the prediction criterion.

Table 4.5: TIMIT Phone Error Rate achieved with different interpolation weights.

Model	Interpolation Weight	Dev	Test
DNN		20.4%	22.2%
LSTM		20.1%	21.7%
PAC-RNN (S)	1.0 (no prediction)	19.9%	21.6%
PAC-RNN (S)	0.8	18.9%	20.0%
PAC-RNN (S)	0.6	19.7%	21.0%

4.4.6 Improved TIMIT baseline

In this subsection, we further improved the baseline system to compare with the state-of-the-art phoneic recognition system on TIMIT. The labels were from a triphone HMM-GMM system from the default Kaldi [79] receipt. Speaker dependent Feature-Space Maximum Likelihood Linear Regression (fMLLR) transforms are applied on top of the MFCC features which is typically used to estimate speaker specific transforms for speaker adaptation.[2] We also delayed the output of LSTM by 5 frames as suggested

[2]fMLLR is also known as Global Constrained Maximum Likelihood Linear Regression (CMLLR) in the literature [25, 78].

Table 4.6: TIMIT Phone Error Rate achieved with different hybrid models.

Model	Dev	Test
DNN [74]	-	20.7
BLSTM [35]	**16.1**	18.0
DNN	19.1	20.6
LSTM	17.8	18.9
PAC-RNN	16.7	17.3
PAC-RNN + LSTM	16.3	**17.2**

in [90] in order to add more context for the LSTM. From Table 4.6, we can observe that now we have a more comparable DNN baseline with [74]. All the gains are consistent with Table 4.1. Our best model was achieved by replacing the correction DNN with a single layer LSTM model. Our model is even comparable with a bidirectional LSTM model in [35] which usually gives 5% to 10% relative gains compared to a unidirectional LSTM model.

4.5 PAC-RNNs for Low-Resource Language Speech Recognition

The PAC-RNN model shows promising results on TIMIT in Section 4.4, but it was unclear whether a similar gain could be achieved on real ASR tasks where the prediction information might already be incorporated into the language model. Here, we successfully apply the PAC-RNN to LVCSR on several low-resource languages. In addition, we study the effect of transfer learning for recurrent architectures. Recurrent networks such as LSTMs [49] are known to require a large amount of training data in order to perform well [90]. For low-resource ASR, multiple groups have incorporated multilingual training in order to alleviate data limitation issues [40, 63, 17]. One popular approach is multi-task training using DNNs. In a multi-task setup, a single DNN is trained to generate outputs for multiple languages with some tied parameters. This approach has been used for robust feature extraction via bottleneck (BN) features [110, 122, 42, 72], or for classifiers in hybrid DNN-HMM approaches [53].

In [60], Karafiát et al. found that using CMLLR transformed BN features as inputs to a hybrid DNN could further improve ASR performance. However, we believe none of this research has investigated recurrent networks for low-resource languages in a multilingual scenario.

In the next following section, we apply PAC-RNNs to low-resource language speech recognition.

4.5.1 Multilingual Systems

Our multilingual system is based on the multilingual framework in [122]. We extract BN features using multilingual networks to train different hybrid neural network architectures.

Stacked bottleneck (SBN) features

The BN features used in this chapter follow Section 2.3.2. An SBN is a hierarchical architecture realized as a concatenation of two DNNs, each with its own bottleneck layer. The outputs from the BN layer in the first DNN are used as the input features for the second DNN, whose outputs at the BN layer are then used as the final features for standard GMM-HMM training.

The inputs of the first layer consist of 23 critical-band energies obtained from a Mel filter-bank. Each of the 23 dimensions are augmented with pitch and probability of voicing [67] and multiplied across time by a Hamming window of length 11 frames. A DCT is then applied for dimensionality reduction. The 0^{th} to 5^{th} coefficients are retained, resulting in a feature of dimensionality $(23+2)*6 = 150$. The input features of the second DNN are the outputs of the BN layer from the first DNN. Unlike [121], speaker dependent fMLLR transforms are also applied to the output of the first BN. Context expansion is done by concatenating frames with time offsets $-10, -5, 0, 5, 10$. Thus, the overall time context seen by the second DNN is 31 frames. Both DNNs use the same setup of 5 hidden sigmoid layers (1024 hidden units) and 1 linear BN layer (80 hidden units). Both of them use tied context-dependent (CD) states as target

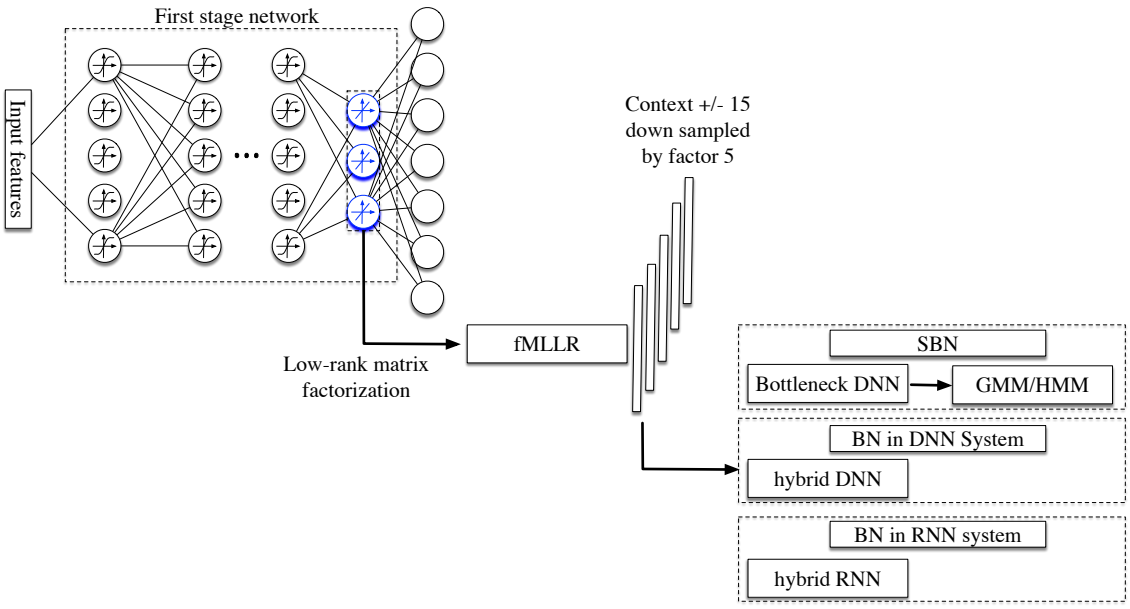


Figure 4-2: We build three different systems using BN-CMLLR features: 1) stacked bottleneck system (SBN), 2) DNN hybrid system using BN features, and 3) RNN hybrid system using BN features.

outputs, which are generated by forced alignment from a GMM-HMM baseline. A final PCA transform is applied to the second set of BN outputs to reduce the number of dimensions to 30. Lastly, delta and delta-delta features are concatenated, resulting in a final dimensionality of 90.

Bottleneck-CMLLR features in a hybrid system

In [60], the authors proposed a DNN hybrid system that used the first stage BN features with speaker adaptation (BN-CMLLR). In this work, we follow a similar approach by replacing the second stage DNN with recurrent architectures (LSTM or PAC-RNN). The BN-CMLLR features were taken from a network trained in a multilingual fashion and adapted to the target language. For the DNN and PAC-RNN, these features were stacked in a context of 31($\pm$15) frames and downsampled by a factor of 5. Following [90], no context expansion is used for the LSTM. The output state label is also delayed by 5 to utilize the information from the future. As illustrated in Figure 4-2, we compare three different systems: SBN, BN in DNN, and

BN in RNN.

Multilingual training and adaptation of SBN features

There are several methods for training multilingual DNNs. In [63, 40], a multilingual phoneset is created, and all the phonemes from the source languages are mapped to the set. The work in [40, 112] shows that a simpler scheme of concatenating each language's outputs in the softmax layer can perform just as well. Furthermore, when concatenating language outputs, normalizing the softmax layer individually within each language during training will yield slightly better results [42, 38]. In this work, we use this method for training the multilingual DNN since it does not require mapping of the phonesets and still provides state-of-the-art results.

When adapting the multilingual DNN to a new language, i.e., the target language, there often exists a limited amount of training data in that language. Adapting the multilingual DNN using data from the target language gives an additional gain over the purely multilingual DNN. For hierarchical architectures, such as the SBN, our previous work in [122] and an independent investigation in [41] seem to suggest that the two DNNs in the SBN architecture behave differently in terms of adaptation. The first DNN extracts more language independent cues from the acoustics, while the second DNN is more language dependent and is more phonetically oriented. As a consequence, our previous work [122] shows that using just the language closest to the target language from the pool of source languages to train the second DNN can serve as a better initialization model than the multilingual second DNN. The closest language can be identified from just the acoustic data by training a Language Identification (LID) system.

A flowchart of how an LID-based multilingual SBN-based ASR system can be trained is shown in Fig. 4-3. We start by adapting the first DNN with data from the target language. Instead of using the second multilingual DNN to initialize, we train the second DNN from random initialization using the closest language's data and output targets. After the DNN converges, we then do a final adaptation to the target language.

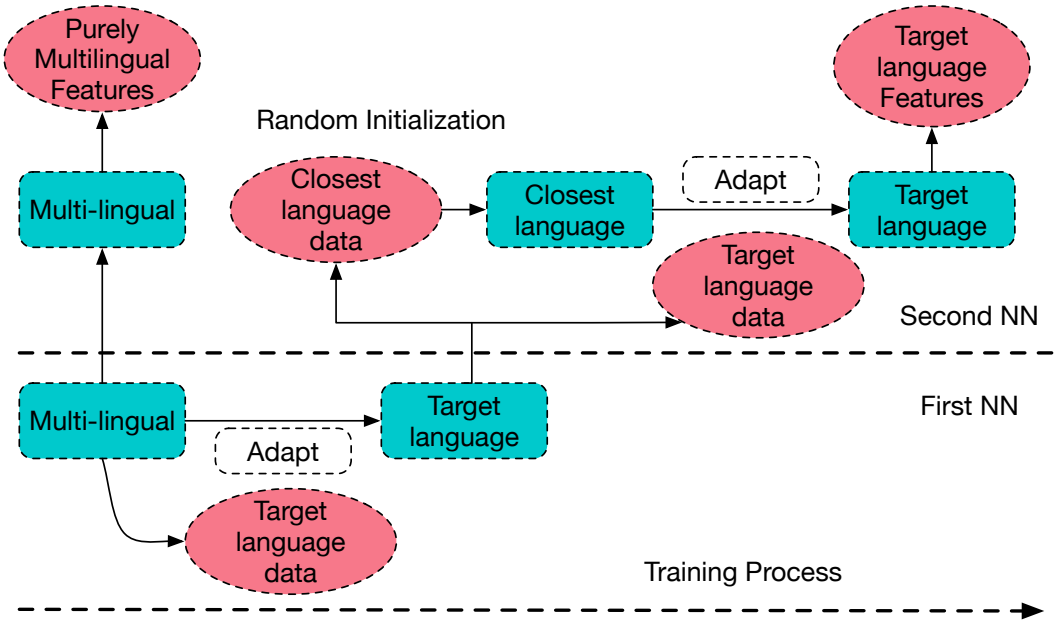


Figure 4-3: Steps to adapt a multilingual SBN to a target language via the closest language selected via LID. The first DNN is adpated from a multilingual first DNN. BN features are extracted from the adapted DNN and used to train a new NN on the closest language (selected by LID) from random initialization. The NN is finally adapted to the target language.

Multilingual training of BN-hybrid system

The input of the hybrid ASR system (DNN, LSTM or PAC-RNN) is the same as the second DNN in the SBN system. During the adaptation stage, the softmax is replaced by the target language state labels (phone labels for the PAC-RNN prediction model) with random initialization while the hidden layers are initialized from the DNN, LSTM or PAC-RNN which is trained using the closest language.

4.5.2 Recognition system

For each language, we used tied-state triphone CD-HMMs, with 2500 states and 18 Gaussian/state. Grapheme-based dictionaries were used for the target languages. Note that for the languages we used from *Babel*, the difference between phonetic and graphemic systems in WER are often less than 1% [65, 26]. All the output targets were from CD states. We use 11 languages (FLP) as the source languages, namely Cantonese, Vietnamese, Tagalog, Pashto, Turkish, Bengali, Assamese, Zulu, Tamil, Haitian, and Lao. To train the multilingual SBN, we kept only the SIL frames that appear 5 frames before and after actual speech. This reduced the total amount of frames for the multilingual DNN to around 520 hours. We observed no loss in accuracy from doing so, and it also reduced the training time significantly. Discriminative training was done on the CD-HMMs using sMBR criterion [62]. The web data was cleaned and filtered using techniques described in [120]. For language modeling, n-gram LMs were created from training data transcripts and the web data. The LMs were then combined using weighted interpolation. The vocabulary included all words that appeared in the training transcripts, augmented with the top 30k most frequent words from the web. We chose 30k words by looking at the rate of out-of-vocabulary (OOV) reduction as we augmented the train vocabulary with frequent words from the web. We report results on the 10-hour development set.

We consider two baseline hybrid systems: a DNN with three 1,024-unit hidden layers, and a stacked LSTM with three layers each containing 512 cells. No gains were observed by further increasing the model size of these baseline systems.

In the PAC-RNN model, the prediction DNN has a 2,048-unit hidden layer and a 80-unit bottleneck layer. For the correction model, we have two systems: a DNN with two 2048-unit hidden layers, or an LSTM with 1024 memory cells. The correction model's projection layer contains 500 units.

All models are randomly initialized without either generative or discriminative pretaining. No momentum is used for the first epoch and a momentum of 0.9 is used for all subsequent epochs. To train the DNN, a learning rate of 0.1 per mini-batch is used for the first epoch and then increased to 1.0 at the second epoch, after which it is kept the same until the development set training criterion no longer improves, under which condition the learning rate is halved. A similar schedule is used to train the LSTMs and PAC-RNNs except that all the learning rates are reduced to 1/10 of that used in the DNN training.

We implemented the hybrid models using the computational network toolkit (CNTK) [119]. The truncated BPTT [115] is used to update the model parameters. We speed up the training as in Chapter 3. For decoding, we fed the posteriors generated by CNTK into the Kaldi ASR toolkit [79], which then generates the recognition results.

4.5.3 PAC-RNN results with BN features

Table 4.7 summarizes the WERs achieved with different models evaluated in this study. The first three rows are the results from SBN systems. Both the multilingual and the closest language systems are adapted to the target language for the whole stacked network. For the hybrid systems, the input is the BN features extracted from the first DNN of the adapted multilingual SBN.

The DNN hybrid system outperforms the multilingual SBN but is very similar to the closest language system. The LSTM improves upon the DNN by around 1%. The PAC-RNN-DNN outperforms LSTM by another percent across all languages. By simply replacing the correction model with a single layer LSTM, we observe further improvements.

Table 4.7: WER (%) results for each ASR system. SBN is the stacked bottleneck system. The closest language of Cebuano, Kurmanji and Swahili are Tagalog, Turkish and Zulu respectively based on our LID experiments.

Target language	Cebuano	Kurmanji	Swahili
Closest language	Tagalog	Turkish	Zulu
SBN models			
Monolingual	73.5	86.2	65.8
Adapted multilingual	65.0	75.5	54.9
Closest language	63.7	75.0	54.2
Hybrid models			
DNN	63.9	74.9	54.0
LSTM	63.0	74.0	53.0
PACRNN-DNN	62.1	72.9	52.1
PACRNN-LSTM	**60.6**	**72.5**	**51.4**
Hybrid models with closest language initialization			
DNN	62.7	73.1	52.4
LSTM	61.3	72.5	52.2
PAC-RNN-DNN	60.8	71.8	51.6
PAC-RNN-LSTM	**59.7**	**71.4**	**50.4**

4.5.4 Effect of transfer learning on recurrent architectures

In this subsection we investigate the effect of the multilingual transfer learning for each model. We first use the rich resource closest language (based on the LID prediction shown in the table) to train DNN, LSTM and PAC-RNN models, and then adapt them to the target language. The lower part of Table 4.7 summarizes the ASR results. As shown, the LSTM models perform significantly better than the baseline SBN system. Using the PAC-RNN model yields a noticeable improvement over the LSTM. Similarly, the PAC-RNN-LSTM can further improve the results.

4.6 Summary

In this chapter, we explored a PAC-RNN model for low-resource language speech recognition. The results on multiple languages demonstrated that the PAC-RNN achieves better performance than DNNs and LSTMs. We also showed that by re-

placing the correction model in the PAC-RNN with an LSTM could further enhance the model. Moreover, the multilingual experiment results show that traditional DNN transfer learning approaches can also be applied to the PAC-RNN architecture.

We believe that next-generation ASR systems will be described as dynamic systems that incorporate many connected components and recurrent feedback, and constantly makes predictions, corrections, and adaptations. For example, the system should be able to automatically identify multiple speakers in a mixed speech setting, and then focus on a specific speaker by ignoring other speakers and noises. We believe the PAC-RNN is the first step towards this direction. However, the first challenge to further extend this framework on large scale dataset such as SWBD was the gradient vanishing problem because the multi-subnetwork further increased the depth of the network. We will try to address this issue in the next chapter.

5

Optimized Structure to Ease Gradient-based Training

5.1 Introduction

In the previous chapter, we described how we could add contextual information for acoustic modeling using the recurrent neural network. As an extension of deep RNNs, the multiple component/feedback loop presented in this thesis increased modeling power, but also makes optimization was difficult due to the multiple layers of each sub-network. This optimization issue arises because training deeper networks are not as straightforward as simply adding layers due to the vanishing gradient problem. Optimization of deep networks has proven to be considerably more challenging [50]. Research on initialization schemes, techniques for training networks in multiple stages, or with temporary companion loss functions attached to some of the layers, have been proposed to solve the vanishing gradient problem. However, these methods require significant amounts of heuristics and hyperparameters tuning.

To address these issues, we present several novel architectures that enable the training of networks with virtually arbitrary depth. The fundamental idea is to use a learned gating mechanism for regulating information flow across different layers. This gating mechanism, called highway connections, enables information flow across several layers/components without attenuation. They alleviate the vanishing gradient problem and enable deeper networks of LSTM RNNs.

We first propose a highway LSTM model that adds "highway" connections between different LSTM layers' cells. This model can be improved by using dropout to control the input of the highway connection. Inspired by the improvement from dropout, we further present two other extensions of the DLSTM RNNs. The grid LSTM (GLSTM) [57] uses separate LSTM blocks along both time and depth axes to improve modeling power. The residual LSTM (RLSTM) RNNs, inspired by linearly augmented DNNs [28], and residual CNNs, [44] contain direct links between lower-layer outputs and higher-layer inputs in DLSTM RNNs. Both the GLSTM and RLSTM RNNs enable us to train deeper models and achieve better accuracy.

5.2 Related Work

After independently developing highway LSTMs, we noticed that similar work had been done in [102, 118, 57]. All of the work shares the same idea of adding gated linear connections between different layers. The highway networks proposed in [102] adaptively carry some dimensions of the input directly to the output so that information can flow across layers much more easily. However, their formulation is different from ours, and their focus is on DNNs. The work in [118] shares the same idea and model structure, while [57] is more general, and uses a generic form. However, their task is based on text, e.g., machine translation, while our focus is on speech recognition. Therefore, we also extend the work in [57] to the speech recognition task.

5.3 Model Description

It is reported in [90] that deep LSTM (DLSTM) RNNs help improve generalization and often outperform single-layer LSTM RNNs. However, when the networks become deeper or more complex, a degradation in accuracy is often observed. Such degradation is not caused by overfitting [44] since the degradation also happens on the training set. For example, in many ASR tasks, 3 to 5 LSTM layers are optimal in a deep LSTM. However, further increasing the depth leads to higher WER. There are

two possible solutions to this degradation problem:

1. to pretrain the network layer by layer, or,

2. to modify the network structure so that can be more easily and effectively optimized.

In this section, we focus on the second approach and propose several architectures that can directly feed information from lower layers to higher layers.

5.3.1 Highway Long Short-term Memory RNNs

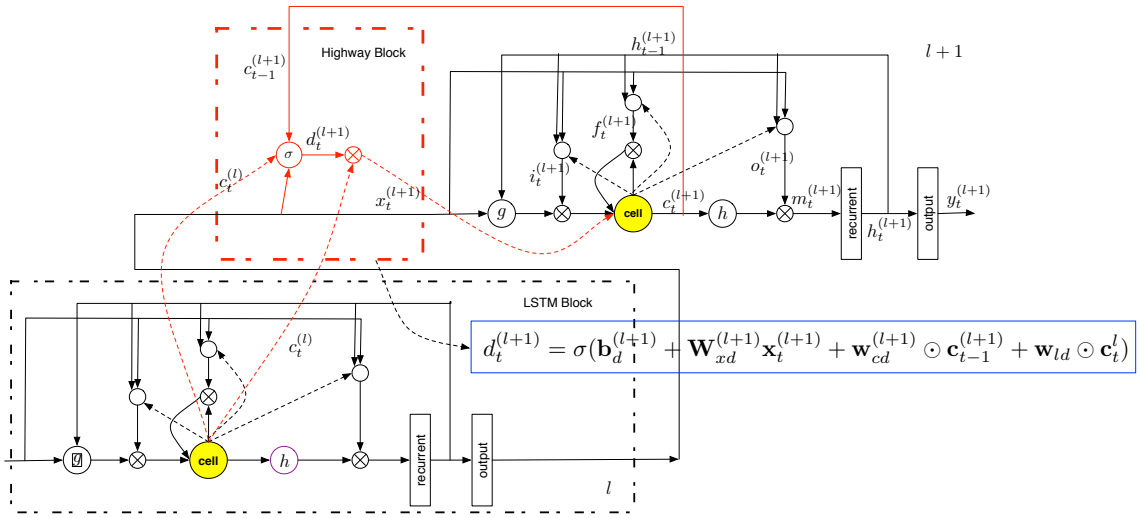


Figure 5-1: Highway Long Short-Term Memory RNNs. We add one more gate (as illustrated in the red dashed-block), d_t^{l+1}, in (l+1)-layer to control how much information can flow from the lower-layer cells.

The Highway LSTM (HLSTM) [124], as illustrated in Figure 5-1, improves upon DLSTM RNNs. It has a direct connection (in the dotted red block) between the memory cells $\mathbf{c}_t^l$ in the lower layer l, and the memory cells $\mathbf{c}_t^{l+1}$ in the upper layer $l+1$. The carry gate controls how much information can flow from the lower-layer cells directly to the upper-layer cells. The gate function at layer $l+1$ at time t is

$$\mathbf{d}_t^{(l+1)} = \sigma(\mathbf{b}_d^{(l+1)} + \mathbf{W}_{xd}^{l+1}\mathbf{x}_t^{(l+1)} + \mathbf{w}_{cd}^{l+1} \odot \mathbf{c}_{t-1}^{(l+1)} + \mathbf{w}_{ld}^{(l+1)} \odot \mathbf{c}_t^{l}), \tag{5.1}$$

where $\mathbf{b}_d^{(l+1)}$ is a bias term, $\mathbf{W}_{xd}^{(l+1)}$ is the weight matrix connecting the carry gate to the input of this layer, $\mathbf{w}_{cd}^{(L+1)}$ is a weight vector from the carry gate to the past cell state in the current layer, $\mathbf{w}_{ld}^{(L+1)}$ is a weight vector connecting the carry gate to the lower layer memory cell, and $\mathbf{d}^{(l+1)}$ is the carry gate activation vectors at layer $l+1$.

Using the carry gate, an HLSTM RNN computes the cell state at layer $(l+1)$ according to

$$\begin{aligned}\mathbf{c}_t^{l+1} &= \mathbf{d}_t^{(l+1)} \odot \mathbf{c}_t^{l} + \mathbf{f}_t^{(l+1)} \odot \mathbf{c}_{t-1}^{(l+1)} \\ &+ \mathbf{i}_t^{(l+1)} \odot \tanh(\mathbf{W}_{xc}^{(l+1)}\mathbf{x}_t^{(l+1)} + \mathbf{W}_{hc}^{(l+1)}\mathbf{h}_{t-1}^{(l+1)} + \mathbf{b}_c),\end{aligned} \tag{5.2}$$

while all other equations are the same as those for the standard LSTM RNNs as described in Eq. (2.10),(2.11),(2.13), and (2.14).

Conceptually, the highway connection is a multiplicative modification that is analogies to the forget gate. Depending on the output of the carry gates, the highway connection smoothly varies its behavior between that of a plain LSTM layer (no connection) and that of direct linking (i.e., passing the cell memory from the previous layer directly, without attenuation). The highway connection between cells in different layers makes the influence of cells in one layer on the other layer more direct and can alleviate the vanishing gradient problem when training deeper LSTM RNNs.

5.3.2 Residual LSTM RNNs

The concept of a residual network was proposed in [44], and is a special case of the linearly augmented model described in [28]. It defines a building block

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{W}_i) + \mathbf{x} \tag{5.3}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers considered. In this study, we replace the convolutional and rectifier linear layer with an LSTM block as

residual LSTM (RLSTM):

$$\mathbf{x}_{l+1} = \text{LSTM}^r(\mathbf{x}_l) + \mathbf{x}_l \tag{5.4}$$

Here r indicates how many layers we want to skip and x_l is the input to l-th LSTM residual block. In [44], it was reported that it is important to skip more than one layer. However, in our study we didn't find it to be necessary.

5.3.3 Grid Long Short-Term Memory RNNs

The grid LSTM (GLSTM) RNN was first introduced in [57]. Unlike traditional LSTM RNN models, which organize LSTM blocks as a temporal chain, grid LSTM RNN models arrange LSTM blocks into multidimensional grids such that each grid contains one set of LSTM blocks for each dimension, including the depth dimension. This architecture introduces per-dimension gated linear dependencies between adjacent cell states, which mitigates the vanishing gradient problem along all dimensions.

Here we consider a two-dimensional grid LSTM model for acoustic modeling, which has time and depth dimensions respectively, as illustrated in Figure 5-2. The computations in each grid are defined as follows:

$$\mathbf{x}_{t,l} = [\mathbf{h}^D_{t,l-1}; \mathbf{h}^T_{t-1,l}] \tag{5.5}$$

$$(\mathbf{h}^T_{t,l}, \mathbf{c}^T_{t,l}) = \textsc{Time}\text{-LSTM}(\mathbf{x}_{t,l}, \mathbf{c}^T_{t-1,l}, \Theta^T) \tag{5.6}$$

$$(\mathbf{h}^D_{t,l}, \mathbf{c}^D_{t,l}) = \textsc{Depth}\text{-LSTM}(\mathbf{x}_{t,l}, \mathbf{c}^D_{t,l-1}, \Theta^D) \tag{5.7}$$

Note that we slightly change the notation here by using subscripts to denote both time and depth, and by using superscripts to indicate a specific set of LSTM blocks. For example, $c^i_{t,l}$ and $h^i_{t,l}$ are cell state, and cell output, respectively, at time t and layer l of i-LSTM, while Θ^i denotes all the parameters of i-LSTM. The cell output of DEPTH-LSTM at the last layer, $h^D_{t,L}$, is passed to the softmax layer for classification.

One last thing that needs to be addressed is $c^D_{t,0}$, for which the value is undetermined. The easiest solution would be to set the value to zero, which would give a flat

initialization of cell states regardless of the input value. Instead, we apply a linear transform such that

$$\mathbf{c}_{t,0}^{D} = \mathbf{V}\mathbf{h}_{t,0}^{D} \tag{5.8}$$

which achieves better performance empirically.

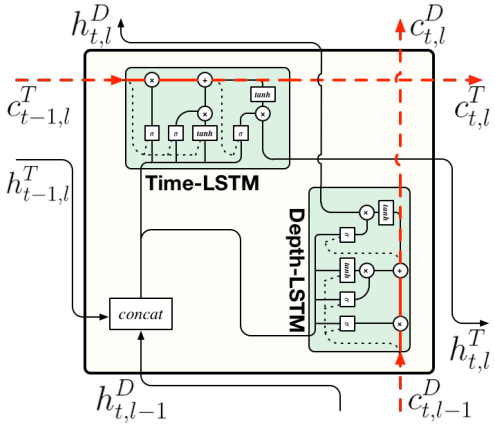


Figure 5-2: The Grid LSTM arranges LSTM blocks into multidimensional grids such that each grid contains one set of LSTM blocks for each dimension.

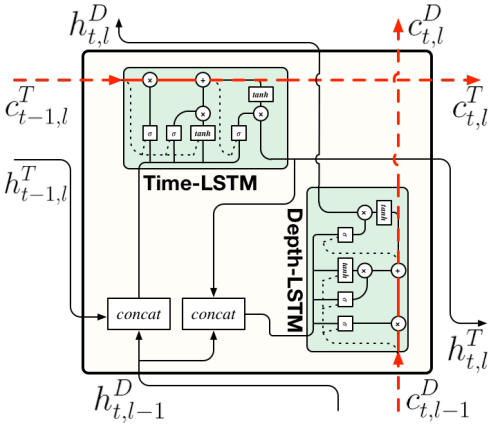


Figure 5-3: In the prioritized Grid LSTM, the input to Depth-LSTM is updated after Time-LSTM of the same grid is processed.

5.3.4 Prioritized Grid Long Short-Term Memory RNNs

In Equation 5.7, we can observe that the cell output from Time-LSTM is not being utilized for classification of the current time step. In other words, we would like the depth dimension to know the output from other dimensions in the current grid so that it is implicitly deeper regarding the number of transformations before being used for classification. To achieve this objective, we slightly modify the formulation

from Section 5.3.3 as follows:

$$\mathbf{x}^T_{t,l} = [\mathbf{h}^D_{t,l-1}; \mathbf{h}^T_{t-1,l}] \tag{5.9}$$

$$(\mathbf{h}^T_{t,l}, \mathbf{c}^T_{t,l}) = \textsc{Time}\text{-LSTM}(\mathbf{x}^T_{t,l}, \mathbf{c}^T_{t-1,l}, \Theta^T) \tag{5.10}$$

$$\mathbf{x}^D_{t,l} = [\mathbf{h}^D_{t,l-1}; \mathbf{h}^T_{t,l}] \tag{5.11}$$

$$(\mathbf{h}^D_{t,l}, \mathbf{c}^D_{t,l}) = \textsc{Depth}\text{-LSTM}(\mathbf{x}^D_{t,l}, \mathbf{c}^D_{t,l-1}, \Theta^D) \tag{5.12}$$

where the input to Depth-LSTM is updated after Time-LSTM of the same grid is processed. We call this model the prioritized grid LSTM (pGLSTM), and illustrate it in Figure 5-3. As opposed to non-prioritized grid LSTM (npGLSTM) in Figure 5-2, $\mathbf{h}^D_{t,l-1}$ is concatenated with updated hidden output, $\mathbf{h}^T_{t,l}$, from the Time-LSTM, and then, fed to the Depth-LSTM. We believe this prioritized version can better utilize the information from all the time steps. Experiments in Section 5.4.6 compare these two different grid LSTMs.

5.3.5 Bidirectional LSTM RNNs

The unidirectional LSTM RNNs we just described can only exploit past history. In speech recognition, however, future contexts also carry information, and should be utilized to enhance the acoustic models further. Bidirectional RNNs take advantage of both past and future contexts by processing the data in both directions with two separate hidden layers. It was shown in [36, 35, 14] that bidirectional LSTN RNNs can indeed improve speech recognition results. In this section, we also extend HLSTM RNNs from unidirectional to bidirectional. Note that the backward layer follows the same equations as used for the forward layer except that $t-1$ is replaced by $t+1$ to exploit future frames and the model operates from $t = T$ to 1. The output of the forward and backward layers are concatenated to form the input to the next layer.

A vanilla BLSTM RNN has latency issues because it requires the entire input sequence at the decoding stage. Moreover, training a BLSTM is very slow, as described in Section 3.4.4. Therefore, we apply the latency-controlled technique we developed

in Section 3.4.4 to all the bidirectional versions of the network in this Chapter.

As described in Section 2.3.4, LSTM with a projection layer (LSTM(P)) can reduce the computation but without performance degradation. Therefore, for all the LSTM layer in the following section, we insert a projection layer to reduce the dimension.

5.4 Experiments

The performance of various models are evaluated using word error rate (WER) in percent below. For the experiments conducted on the AMI meeting corpus, the SDM eval set is used if not specified otherwise. Since we do not exclude overlapping speech segments during model training, in addition to results on the full eval set, we also show results on a subset that only contains non-overlapping speech segments, as in [104].

5.4.1 Dataset

Our experiments to study the behavior of different LSTM variants were based on five different speech corpora: AMI, HKUST, GALE Mandarin, Switchboard and Arabic MGB. These corpora span a wide variety of configurations, ranging from 100 hours to 1200 hours, include three languages, multiple accents, and recording under different scenarios and sampling rates. For more details, readers are invited to refer to Section 2.4.

5.4.2 Model Setup

We compared four RNN models: (1) LSTMP (baseline) (2) HLSTMP, (3) RLSTMP, and (4) GLSTMP. For all the models, we follow the configurations reported in our paper [124]. Each layer contains 1024 memory cells, and a 512-node linear projection layer is added on top of each layer's output. For RLSTM, we add shortcut connections from each layer's input to its output.

We first compare HLSTM with the baseline LSTM over different configurations,

e.g., the number of layers on the AMI corpus to prove it can help train a deeper network. Then we further evaluate more highway variants (RLSTM and GLSTM) on different scale tasks (from 100hrs to 1200hrs). For our non-prioritized and prioritized grid LSTM models (npGLSTM/pGLSTM), we chose the same configuration for both time-LSTM and depth-LSTM as the baseline models.

5.4.3 Training

We use Kaldi [79] for feature extraction, decoding, and training of initial HMM-GMM models. Maximum likelihood-criterion context-dependent speaker adapted acoustic models with Mel-Frequency Cepstral Coefficient (MFCC) features are trained with standard Kaldi recipes. Forced alignment is performed to generate frame-level labels for neural network acoustic model training.

The Computational Network Toolkit (CNTK) [119] is used for the remaining neural network training. As [89] suggests, all weights are randomly initialized from the uniform distribution with range $[-0.05, 0.05]$, and all biases are initialized to 0 without generative or discriminative pretraining [92]. All neural network models, unless explicitly stated otherwise, are trained with a cross-entropy (CE) criterion, using truncated back-propagation-through-time (BPTT) [115] for optimization, which unrolls 20 frames and processes 40 utterances in parallel in each mini-batch. No momentum is used for the first epoch, and a momentum of 0.9 is used for subsequent epochs [123]. $L2$ constraint regularization [48] with a weight of 10^{-5} is applied.

For *AMI*, *HKUST*, *SWBD* and *GALE Mandarin*, ten percent of the training data is held out as a validation set, which is used to control the learning rate. When no gain is observed after an epoch, the learning rate is halved, and the model with the lowest validation loss is reloaded. For *Arabic MGB*, all data is used for training, and the learning rate is halved after each epoch; in addition, we use 4-GPU parallel training with the model-averaging stochastic gradient descent (SGD) method [80]. Specifically, as [13] suggests, we start with a seed model trained with standard SGD method for one epoch to achieve better performance.

To train the bidirectional model, the latency-controlled method described in Sec-

tion 3.4.4 was applied. We set $N_c = 80$ and $N_r = 20$ and also processed 40 utterances simultaneously. To train the recurrent model with the sMBR criterion, we adopted the two-forward-pass method described in Section 3.4.5, and processed 40 utterances simultaneously.

The input features for all models are 80-dimensional log Mel filterbank features computed every 10 ms, with an additional 3-dimensional pitch features, unless explicitly stated. The output targets are context-dependent triphone states, of which the numbers are determined by the last HMM-GMM training stage. Table 5.1 shows the number of output targets in each dataset.

Table 5.1: Number of output targets in each dataset.

	AMI	HKUST	GALE	SWBD	MGB
#states	3943	2825	4198	8802	3711

5.4.4 Highway LSTMP Results

The performance of the HLSTMP structure that can help train deeper networks is evaluated below.

3-layer Highway (B)LSTMP

Table 5.2 gives the WER performance of the 3-layer LSTMP and BLSTMP RNNs, as well as their highway versions, on the AMI corpus. The performance of the DNN network is also listed for comparison. From the table, it is clear that the highway version of the LSTMP RNNs consistently outperforms its non-highway companions, though with a small margin.

Highway (B)LSTMP with dropout

Dropout can be applied to the highway connection to control its flow: a high dropout rate essentially turns off the highway connection, and a small dropout rate, on the other hand, keeps the connection alive. In our experiments, for early training stages,

Table 5.2: Performance of highway (B)LSTMP RNNs on the AMI corpus. SDM setup is adopted.

System	#Layers	with overlap	no overlap
DNN	6	57.5	48.4
LSTMP	3	50.7	41.7
HLSTMP	3	50.4	41.2
BLSTMP	3	48.5	38.9
BHLSTMP	3	48.3	38.5

we use a small dropout rate of 0.1. We increase it to 0.8 after 5 epochs of training. The performance of highway (B)LSTMP networks with dropout is shown in Table 5.3. As we can see, dropout helps to bring down the WER further for highway networks. The gain from dropout suggests that there is better way to control the information flow of the highway connection. This observation motivates us investigate different variants of highway connections such as RLSTMP and GLSTMP in Section 5.3.2 and 5.3.4.

Table 5.3: Performance of highway (B)LSTMP RNNs with dropout on the AMI corpus. SDM setup is adopted.

System	#Layers	with overlap	no overlap
LSTMP	3	50.7	41.7
HLSTMP + dropout	3	49.7	40.5
BLSTMP	3	48.5	38.9
BHLSTMP + dropout	3	**47.5**	**37.9**

Deeper highway LSTMP

When a network goes deeper, the training usually becomes more difficult. Table 5.4 compares the performance of shallow and deep networks. From the table, we can see that for a normal LSTMP network, when it goes from 3 layers to 8 layers, the recognition performance degrades dramatically. For the highway network, however, the WER only increases a little. If we go even deeper, e.g. to 16 layers, normal LSTMP training would diverge but the highway network can still be trained well.

The table suggests that the highway connection between LSTM layers allows the network to go much deeper than the normal LSTM networks. This also indicates that the HLSTMP may gain more when we have much more data since we could train much deeper models.

Table 5.4: Comparison of shallow and deep networks on the AMI corpus. The SDM setup is adopted.

System	#layers	with overlap	no overlap
LSTMP	3	50.7	41.7
LSTMP	8	52.6	43.8
LSTMP	16	N/A	N/A
HLSTMP	3	50.4	41.2
HLSTMP	8	50.7	41.3
HLSTMP	16	50.7	41.2

5.4.5 Grid LSTMP Results

The performance of various models is reported in character error rate (CER) for Chinese corpora, and word error rate (WER) for English and Arabic.

5.4.6 Prioritized/Non-Prioritized Grid LSTM

We first compare the two grid LSTM models along with baseline models on two medium-sized datasets: *HKUST* and *GALE Mandarin*. The CER of the different models are shown in Table 5.5. Both grid LSTM architectures outperform the vanilla LSTM model, as well as the highway LSTM model. Specifically, a 3% to 5% relative gain is achieved for the non-prioritized grid LSTM compared to the vanilla LSTM. The result suggests that grid LSTM models are empirically better solutions for introducing gated linear dependencies across the depth dimension.

Between the two grid architectures, the prioritized grid LSTM model consistently shows better ASR performance, providing an additional 1% relative gain compared to the non-prioritized model. This result supports our hypothesis that the LSTM whose

output is fed into the final softmax layer should be prioritized in order to obtain more recent information.

Table 5.5: CER comparisons between non-prioritized (npGLSTMP) and prioritized Grid LSTMPs (pGLSTMP) on HKUST and GALE.

Model	#layers	HKUST	GALE
LSTMP	3	33.29	23.96
HLSTMP	3	32.86	23.33
npGLSTMP	3	32.32	22.80
pGLSTMP	3	**32.06**	**22.54**

Comparisons with Alternative Deep LSTMs

We compared alternative deep LSTM architectures with the prioritized Grid LSTM on the AMI corpus, when increasing model depth. Table 5.6 shows the detailed results. When increasing the number of layers from 3 to 8, the vanilla LSTM deteriorates significantly; on the other hand, the performance of the HLSTM only degrades slightly and levels off after 8 layers.

In contrast, both the RLSTM and pGLSTM benefit from increasing the depth. The pGLSTM consistently performs better than all the other models, and it is worth noting that the 3-layer pGLSTM model achieves roughly the same accuracy as the 16-layer RLSTM model, while the latter takes much longer to train and much more space. We argue that utilizing the vertical (depth) information with an LSTM is essential for achieving good performance. We were not able to train a pGLSTM model with 16 layers. Careful parameter initialization may be required.

Deeper Prioritized Grid LSTM

We conducted extensive experimentation to verify the effectiveness of the prioritized Grid LSTMP on all four datasets. Table 5.7 summarizes the results of the baseline models as well as the proposed models and includes references to other models tested on the same dataset that is reported in the literature.

Table 5.6: WER comparisons between deep LSTM, HLSTM, RLSTM, and GLSTM models on the AMI corpus.

Model	#layers	#params	with overlap	no overlap
LSTMP	3	12M	50.7	41.7
LSTMP	8	36M	52.6	43.8
HLSTMP	3	14M	50.4	41.2
HLSTMP	8	40M	50.7	41.3
HLSTMP	16	82M	50.7	41.2
RLSTMP	3	12M	51.3	42.0
RLSTMP	8	36M	50.5	40.8
RLSTMP	16	74M	49.9	40.4
pGLSTMP	3	25M	49.8	40.5
pGLSTMP	8	72M	**49.0**	**39.6**

The performance of the pGLSTMP models is consistently superior to the baseline models, with gains being observed when increasing the number of layers from 3 to 5 on all datasets.[1] The 5 layer pGLSTMP models set new state-of-the-art results on the *HKUST*, *SWBD*, and *GALE* datasets. As for *AMI*, the best result is the state-of-the-art uni-directional recurrent model. As in Section 5.4.4, Table 5.3 shows that the bi-directional version gives about 2% absolute error reduction. Therefore, we further modify the pGLSTMP to be a latency-controlled bidirectional pGLSTMP (LC-pBGLSTMP). We can see we got consistent gain over the bidirectional LSTMP (BLSTMP) in the *SWBD* corpus from 11.7% to 10.8%. After sMBR training, our result is 0.2% worse than previously published best number [81]. However, we didn't apply LF-MMI training and use i-vector as additional inputs as suggested in [81]. The GLSTMP can actually replace LSTMP in [81] and we believe further gain can be observed.

[1]For the Grid LSTMP, 5-layer setup is more similar to the 10-layer setup in a regular DLSTMP because we have two LSTMPs in a Grid LSTM.

Table 5.7: WER comparisons on all four datasets using different hybrid models. pGLSTMP is our proposed prioritized grid LSTMP. LC-BLSTMP is the latency-controlled BLSTMP. The number inside the brackets is the results after sMBR training.

Model	#L	HKUST	SWB	GALE	MGB
Stacked maxout LSTMPs [69]	3	33.89	-	-	-
HCLDNN [52]	11	-	-	22.41	-
BLSTMP + LF-MMI [81]	3	-	10.3 (9.6)	-	-
LSTM	3	33.29	12.2	23.96	23.56
HLSTMP	3	32.86	-	23.33	23.32
HLSTMP	5	32.40	-	22.63	23.12
pGLSTMP	3	32.06	12.0	22.54	22.36
pGLSTMP	5	**31.36**	**11.2**	**22.33**	**22.18**
LC-BLSTMP (SMBR)	5	-	11.7(10.5)	-	-
LC-pBGLSTMP (SMBR)	5	-	10.8(9.8)	-	-

Prioritized Grid LSTM with Sequence Training

Finally, we perform sequence training for the prioritized Grid LSTM on the 1200 hour Arabic MGB dataset. Detailed results are shown in Tables 5.8 and 5.7. The results suggest that the prioritized Grid LSTMP model can largely benefit from sequence training. Here an 8.8% relative improvement in WER is observed for sequence training of the 3-layer model, while a slightly larger 9.3% relative improvement is observed in a 5-layer model on *MGB*. In *SWBD*, we get new state-of-the-art results, 9.8%, on the eval2000 test set in Table 5.7.

Table 5.8: Performance of sequence training using pGLSTM on MGB.

Model	#layers	MGB
pGLSTMP	3	22.36
pGLSTMP	5	22.18
pGLSTMP (sMBR)	3	20.40
pGLSTMP (sMBR)	5	**20.11**

5.5 Summary

In this chapter, we have proposed the HLSTM model for ASR, with a focus on deeper structure. Inspired by recent deeper architectures, we also explored the different version of "highway" networks. The experimental results on *AMI* showed that:

- They all outperform traditional DLSTMP and allow us to train a deeper model.
- GLSTMP is the best choice if we do not need to go very deep.
- RLSTMP has more potential in a very deep setup than HLSTMP although it does not perform well in a shallow configuration.

We also applied the highway structure to larger tasks. We found more gains can be observed if we have more layers with more data.

Furthermore, the highway connection could be modified or extended in various ways to other applications or types of data. For example, it can be combined with deep convolutional networks. In the context of PAC-RNN, we can extend it as a plug-in into more complex neural architectures that allow a gated information flow between different neural components such as the prediction model and the correction model in Section 4.3.

6

Deep Neural Networks for End-to-End Speech Recognition

6.1 Introduction

In the previous chapters, we described how one could improve hybrid NN-HMM systems by modifying the network structure. However, as mentioned, traditional hybrid systems are complicated and composed of many individual components: pronunciation models, acoustic models, language models and text normalization. Each component makes various modeling assumptions, which is problematic in that training different modules separately with different criteria is probably not optimal for solving the overall task. In previous chapters, DNN acoustic models are optimized towards frame-level cross entropy, which requires a one-to-one mapping from the input sequence to the output state sequence. The fixed alignment is easy to train. However, it has limited representational power when we consider more advanced neural architectures. For example, the vision community has observed that higher layers in a neural network [66] usually represent higher level abstractions for object detection. But for acoustic modeling, we have to keep the same resolution for each layer. Recent work in this area attempts to rectify this disjoint training issue by designing models that are trained end-to-end – from speech directly to transcripts. In this chapter, we propose a new model in a sequence-to-sequence framework, which is inspired by Chapter 5 and some recent advances from the vision community.

The sequence-to-sequence (seq2seq) model with attention [5] has recently demonstrated a promising new direction for ASR that entirely sidesteps the complicated machinery developed for classical ASR [16, 12, 6, 7, 11]. It is able to do this because it is not restricted by the classical independence assumptions of Hidden Markov Model (HMM) [83] and Connectionist Temporal Classification (CTC) [32] models. As a result, a single *end-to-end model* can jointly accomplish the ASR task within one single large neural network.

The foundational work on seq2seq models, however, has relied on simple neural network encoder and decoder models using recurrent models with LSTMs [6, 11] or GRUs [6]. However, their use of hierarchy in the encoders demonstrates that better encoder networks in the model should lead to better results. In this chapter, we significantly extend the state of the art in this area by developing very deep hybrid convolutional and recurrent models, using recent developments in the vision community.

CNNs [66] have been successfully applied to many ASR tasks [3, 88, 10]. Unlike DNNs [47], CNNs explicitly exploit structural locality in the spectral feature space. CNNs use shared weight filters and pooling to give the model better spectral and temporal invariance properties, thus they typically yield better generalized and more robust models compared to DNNs [85]. Recently, very deep CNNs architectures [98] have also been shown to be successful in ASR [96, 95], using more non-linearities, but fewer parameters. Such a strategy can lead to more expressive models with better generalization.

While very deep CNNs have been successfully applied to ASR, recently there have been several advancements in the computer vision community on very deep CNNs [98, 105] that have not been explored in the speech community. We explore and apply some of these techniques in our end-to-end speech model:

1. Network-in-Network (NiN) [70] increases network depth through the use of 1x1 convolutions. This allows us to increase the depth and expressive power of a network while reducing the total number of parameters that would have been needed otherwise to build such deeper models. NiN has seen great success in

computer vision, building very deep models [105]. We show how to apply NiN principles in hierarchical Recurrent Neural Networks (RNNs) [46].

2. Batch Normalization (BatchNorm) [54] normalizes each layer's inputs to reduce internal covariate shift. BatchNorm speeds up training and acts as a regularizer. BatchNorm has also seen success in end-to-end CTC models [4]. The seq2seq attention mechanism [5] produces a high variance in the gradient (especially with random initialization); without BatchNorm we were unable to train the deeper seq2seq models we demonstrate in this chapter. We extend previous work and show how BatchNorm can be applied to seq2seq acoustic model encoders.

3. Residual Networks (ResNets) [44] learn a residual function of the input through the usage of skip connections. ResNets allow us to train very deep networks without suffering from poor optimization or generalization which typically happens when the network is trapped in local minima. We explore these skip connections to build deeper acoustic encoders.

4. Convolutional LSTM (ConvLSTM) [97] uses convolutions to replace the inner products within the LSTM unit. ConvLSTM allows us to maintain structural representations in our cell state and output. Additionally, it allows us to add more computation to the model while reducing the number of parameters for better generalization. We show how ConvLSTMs can be beneficial and replace LSTMs.

We are driven by the same motivation that led to the success of very deep networks in vision [98, 105, 54, 44] – add depth of processing using more non-linearities and expressive power, while keeping the number of parameters manageable, in effect increasing the amount of computation per parameter. In this chapter, we use very deep CNN techniques to significantly improve over previous shallow seq2seq speech recognition models [6]. Our best model achieves a WER of 10.53%, where our baseline achieves a WER of 14.76%. We present a detailed analysis on how each technique improves the overall ASR performance in the following sections.

6.2 Model

In this section, we will describe the details of each component of our model.

6.2.1 Sequence-to-Sequence Model

Sequence-to-Sequence (seq2seq) [103] model is a framework that encodes an input sequence $\mathbf{x}$ of arbitrary length into some latent representation $\mathbf{h}$, and uses $\mathbf{h}$ to decode an output sequence, $\mathbf{y}$, of arbitrary length. Compared with hybrid NN-HMM approaches, the seq2seq-based approach does not make any conditional independence assumptions, and directly estimates the posterior $p(\mathbf{y}|\mathbf{x})$ based on the chain rule:

$$p(\mathbf{y}|\mathbf{x}) = \prod_t p(\mathbf{y}_t|\mathbf{y}_1, \cdots, \mathbf{y}_{t-1}, \mathbf{x}), \tag{6.1}$$

where $p(\mathbf{y}_t|\mathbf{y}_1, \cdots, \mathbf{y}_{t-1}, \mathbf{x})$ is obtained by

$$p(\mathbf{y}_t|\mathbf{y}_1, \cdots, \mathbf{y}_{t-1}, \mathbf{x}) = \text{Decoder}(\mathbf{h}_{t-1}, \mathbf{y}_{t-1}) \tag{6.2}$$

$$\mathbf{h}_t = \text{Encoder}(\mathbf{x}). \tag{6.3}$$

The encoder, Eq. (6.3), converts input feature vectors $\mathbf{x}$ into a framewise hidden vector $\mathbf{h}_t$ in an encoder network based on BLSTM, i.e., $\text{Encoder}(X) \triangleq \text{BLSTM}(X)$. A decoder network is another recurrent network conditioned on previous output $\mathbf{y}_{t-1}$ and hidden vector $\mathbf{h}_{t-1}$.

Attention

Attention [33] is a mechanism to locate and extract information from a memory source. One implementation of attention is content-based attention [5], where there is a content-based query, $\mathbf{q}$ and a memory source $\mathbf{h}$. The content-based attention mechanism includes three components:

- A $\mathbf{e} = \text{DistanceMetric}(\mathbf{q}, \mathbf{h}$ to computes computes energies between the query, $\mathbf{q}$, and all the elements of the memory source, $\mathbf{h}$.

- A $\alpha = \text{Normalize}(e)$ function to convert $\mathbf{e}$ into an alignment distribution over $\mathbf{h}$.

- A context vector $\mathbf{c}$ based on the alignment distribution α, and the memory source.

In the original implementation of the seq2seq model with attention, the DistanceMetric function is a MLP network, and Normalize is the softmax function. The context vector is weighted sum of all the hidden representation, $\mathbf{c} = \sum_t \alpha_t \mathbf{h}_t$.

The seq2seq model with attention has been applied successfully to many applications, including machine translation [116], grapheme-to-phoneme conversion [117], and speech recognition [12]. We will describe seq2seq model with attention for speech recognition in the next section.

6.2.2 Listen, Attend and Spell

As illustrated in Figure 6-1, Listen, Attend and Spell (LAS) [12] is an attention-based seq2seq model which learns to transcribe an audio sequence to a word sequence, one character at a time. Let $\mathbf{x} = (x_1, \ldots, x_T)$ be the input sequence of audio frames, and $\mathbf{y} = (y_1, \ldots, y_S)$ be the output sequence of characters. The LAS models each character output y_i using a conditional distribution over the previously emitted characters $y_{<i}$ and the input signal $\mathbf{x}$. The probability of the entire output sequence is computed using the chain rule of probabilities:

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|\mathbf{x}, \mathbf{y}_{<\mathbf{i}})$$

The LAS model consists of two sub-modules: the listener and the speller. The listener is an acoustic model encoder and the speller is an attention-based character decoder. The encoder (the Listen function) transforms the original signal $\mathbf{x}$ into a high level representation $\mathbf{h} = (h_1, \ldots, h_U)$ with $U \leq T$. The decoder (the AttendAndSpell func-

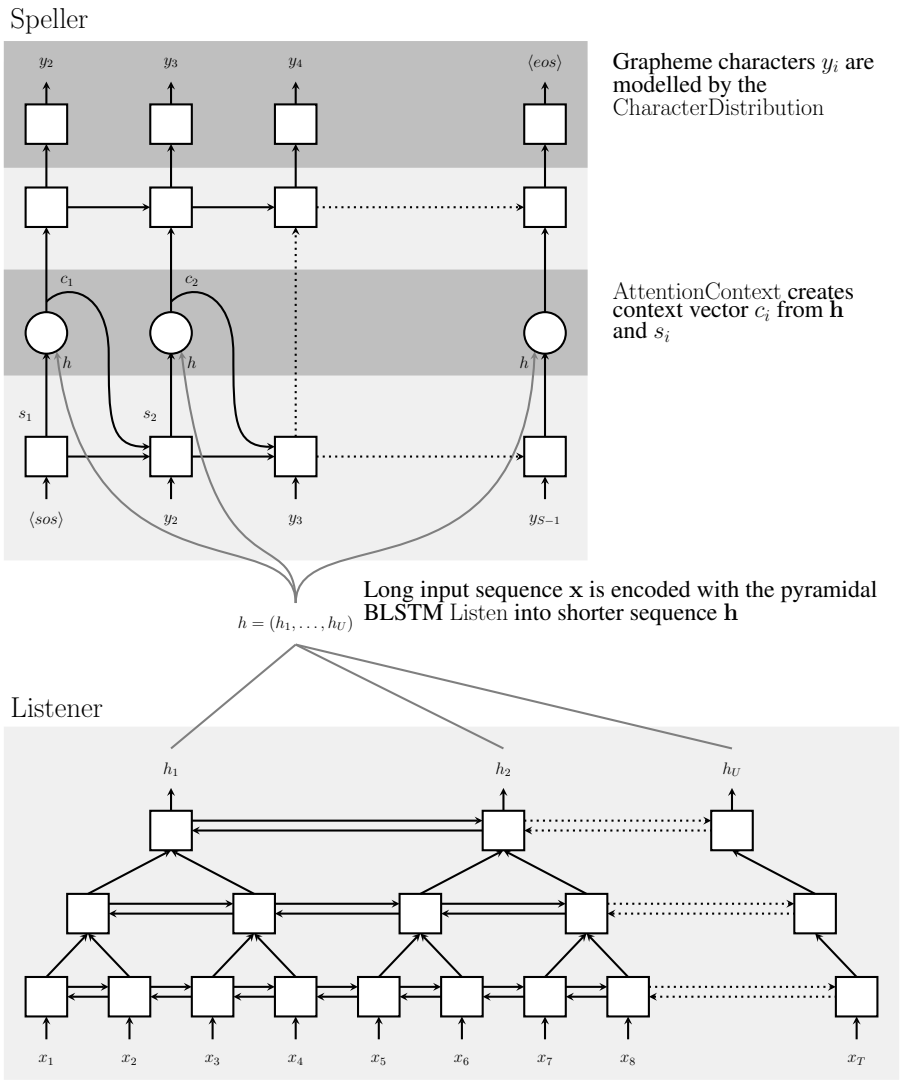


Figure 6-1: Listen, Attend and Spell (LAS) model: the encoder (listerner) is a pyramidal BLSTM encoding the input sequence $\mathbf{x}$ into hidden representation $\mathbf{h}$, the decoder (speller) is an attention-based decoder generating the characters from $\mathbf{h}$. This figure is copied from [12].

tion) consumes $\mathbf{h}$ and produces a probability distribution over character sequences:

$$\mathbf{h} = \text{Listen}(\mathbf{x}) \tag{6.4}$$

$$P(\mathbf{y}|\mathbf{x}) = \text{AttendAndSpell}(\mathbf{h}) \tag{6.5}$$

The LAS Listen function is a stacked Bidirectional Long-Short Term Memory (BLSTM) [35] network with hierarchical subsampling, as described in [12]. In our work, we replace it with a network of very deep CNNs and BLSTMs. The AttendAndSpell is an attention-based transducer [5], which generates one character y_i at a time:

$$s_i = \text{DecodeRNN}([y_{i-1}, c_{i-1}], s_{i-1}) \tag{6.6}$$

$$c_i = \text{AttentionContext}(s_i, \mathbf{h}) \tag{6.7}$$

$$p(y_i|\mathbf{x}, \mathbf{y}_{<\mathbf{i}}) = \text{TokenDistribution}(s_i, c_i) \tag{6.8}$$

The DecodeRNN function produces a transducer state s_i as a function of the previously emitted token y_{i-1}, the previous attention context c_{i-1}, and the previous transducer state s_{i-1}. In our implementation, DecodeRNN is an LSTM [49] function without peephole connections.

The AttentionContext function generates c_i with a content-based Multi-Layer Perceptron (MLP) attention network [5]. Energies e_i are computed as a function of the encoder features $\mathbf{h}$ and current transducer state s_i. The energies are normalized into an attention distribution α_i. The attention context c_i is then created as an α_i weighted linear sum over $\mathbf{h}$:

$$e_{i,j} = \langle v, \tanh(\phi(s_i, h_j)) \rangle \tag{6.9}$$

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{j'} \exp(e_{i,j'})} \tag{6.10}$$

$$c_i = \sum_j \alpha_{i,j} h_j \tag{6.11}$$

where ϕ is linear transform. The TokenDistribution function is an MLP function with softmax outputs modelling the character distribution $p(y_i|\mathbf{x}, \mathbf{y}_{<i})$.

This thesis will focused on the encoder part of seq2seq model.

6.2.3 Network in Network

In our study, we add depth through NiN modules in the hierarchical subsampling connections between LSTM layers. We introduce a projected subsampling layer, wherein we simply concatenate two time frames to a single frame, project into a lower dimension and apply BatchNorm and a rectified linear unit (ReLU) non-linearity to replace the skip subsampling connections in [12]. Moreover, we further increase the depth of the network by adding more NiN 1×1 convolution modules in between each LSTM layer.

6.2.4 Convolutional Layers

Unlike fully connected layers, Convolutional Neural Networks (CNNs) take into account the input topology and are designed to reduce translational variance by using weight sharing with convolutional filters. CNNs have shown improvement over traditional fully-connected deep neural networks on many ASR tasks [85, 10]. We investigate the effect of convolutional layers in seq2seq models.

We can treat the filter-bank feature of one utterance as an image and do convolutions over it. Let input feature $\mathbf{x} \in \mathbb{R}^{T_\mathbf{x} \times F_\mathbf{x}}$ be a two dimensional matrix, where $T_\mathbf{x}$ denotes the context window width and $F_\mathbf{x}$ denotes the number of frequency bands. Suppose there are K kernels with weight $\mathbf{W}_1$, $\mathbf{W}_2$, $\cdots$, $\mathbf{W}_K$ and bias b_1, b_2, $\cdots$, b_K. We use k to index kernels and the k-th kernel $\mathbf{W}_k \in \mathbb{R}^{T_k \times F_k}$. The activation (also called a feature map) of the k-th kernel centered at the (t, f)-position of the input feature is

$$h_{k_{t,f}} = \theta(\sum_{i=1}^{T_k} \sum_{j=1}^{F_k} x_{i+(t-\lceil \frac{T_k}{2} \rceil),j+(t-\lceil \frac{F_k}{2} \rceil)} W_{k_{i,j}} + b_k), \tag{6.12}$$

where θ is the activation function, which we set to be rectified linear units here. Note that we set $x_{i',j'} = 0$ if i', j' exceeds the boundary.

In a hybrid system, convolutions require the addition of a context window for each frame, or a way to treat the full utterance as a single sample [95]. One advantage of the seq2seq model is that the encoder can compute gradients over an entire utterance all

at once. Moreover, strided convolutions are an essential element of CNNs. For LAS, applying striding is also a natural way to reduce temporal resolution. For example, when the stride is 2 then the filters jump 2 frequency bands (similar to "pixel" in a image) at a time as we slide them around. This will produce smaller output volumes spatially.

6.2.5 Batch Normalization

Batch normalization (BatchNorm) [54] is a technique to accelerate training and improve generalization, which is widely used in the computer vision community. Given a layer with output $\mathbf{x}$, BatchNorm is implemented by normalizing each layer's inputs:

$$\text{BatchNorm}(\mathbf{x}) = \gamma \frac{\mathbf{x} - \text{E}[\mathbf{x}]}{(\text{Var}[\mathbf{x}] + \epsilon)^{\frac{1}{2}}} + \beta \tag{6.13}$$

where γ and β are learnable parameters. The standard formulation of BatchNorm for CNNs can be readily applied to DNN acoustic models and cross-entropy training. For our seq2seq model, since we construct a minibatch containing multiple utterances, we follow the sequence-wise normalization [4]. For each output channel, we compute the mean and variance statistics across all timesteps in the minibatch.

6.2.6 The Convolutional LSTM model

The Convolutional LSTM (ConvLSTM) was first introduced in [97]. Although the fully connected LSTM layer has proven powerful for handling temporal correlations, it cannot maintain structural locality and is more prone to overfitting. ConvLSTM is an extension of FC-LSTM which has convolutional structures in both the input-to-state

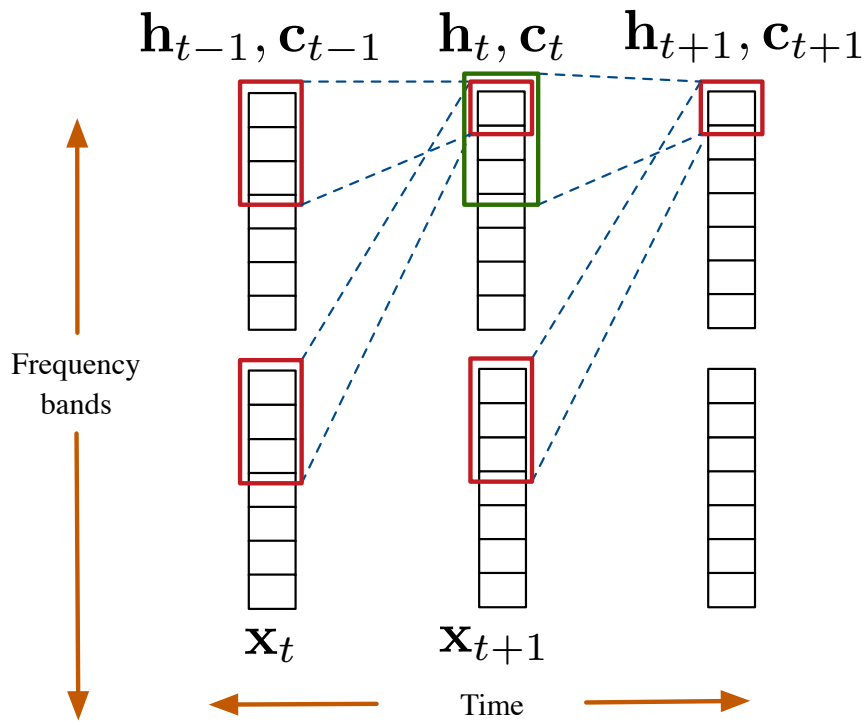


Figure 6-2: The Convolutional LSTM (ConvLSTM) maintains spectral structural locality in its representation. We replace the inner product of the LSTM with convolutions.

and state-to-state transitions:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi} * \mathbf{x}_t + \mathbf{W}_{hi} * \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{xf} * \mathbf{x}_t + \mathbf{W}_{hf} * \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc} * \mathbf{x}_t + \mathbf{W}_{hc} * \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo} * \mathbf{x}_t + \mathbf{W}_{ho} * \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{6.14}
$$

We apply these equations iteratively from $t = 1$ to $t = T$, where $\sigma(*)$ is the logistic sigmoid function, $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t, \mathbf{c}_t$ and $\mathbf{h}_t$ are vectors to represent values of the input gate, forget gate, output gate, cell activation, and cell output at time t, respectively. $\odot$ denotes element-wise product of vectors. $\mathbf{W}_*$ are the convolutional filter matrices con-

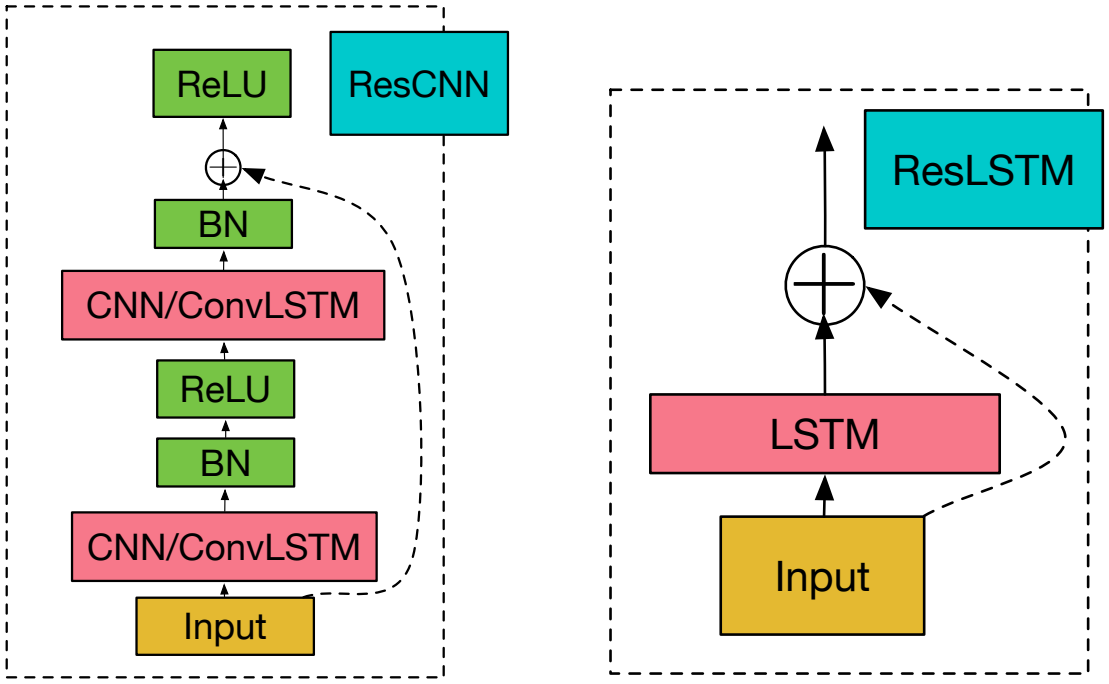
necting different gates, and $\mathbf{b}_*$ are the corresponding bias vectors. The key difference is that $*$ is now a convolution, while in a regular LSTM $*$ is a matrix multiplication. Figure 6-2 shows the internal structure of a convolutional LSTM. The state-to-state and input-to-state transitions can be achieved by a convolutional operation (here we ignore the multiple input/output channels). To ensure the attention mechanism can find the relation between encoder output and the test embedding, FC-LSTM is still necessary. However, we can use these ConvLSTMs to build deeper convolutional LSTM networks before the FC-LSTM layers. We expect this type of layer to learn better temporal representations compared to purely convolutional layers while being less prone to overfitting than fully-connected-LSTM layers. We found bidirectional convolutional LSTMs to consistently perform better than unidirectional layers. All experiments reported in this chapter used bidirectional models; from here on we use convLSTM to mean bidirectional convLSTM.

6.2.7 Residual Network

Deeper networks usually improve generalization and often outperform shallow networks. However, they tend to be harder to train and slower to converge when the model becomes very deep. Several architectures have been investigated in Chapter 5 to enable training of very deep networks. As we mentioned in Chapter 5, the idea behind these approaches is similar to the LSTM innovation – the introduction of linear or gated linear dependence between adjacent layers in the NN model to solve the vanishing gradient problem. One difference compared to Chapter 5 is that deep LSTM models are very easy to overfit in a Seq2Seq model as we shown in Section 6.3.3, compared to a hybrid NN-HMM system. Instead, we use a residual CNN, to train deeper networks.

A residual network [44] contains direct links between the lower layer outputs and the higher layer inputs. It defines a building block:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{W}_i) + \mathbf{x} \tag{6.15}$$



(a) ResCNN block (b) ResLSTM block

Figure 6-3: Residual block for different layers. ResCNN is a CNN block with CNN or ConvLSTM, Batch Normalization (BatchNorm) and ReLU non-linearities. The ResLSTM is a LSTM block with residual connections.

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers considered. The function $\mathcal{F}$ can be one or more convolutional or convLSTM layers. The residual block for different layers is illustrated in Figure 6-3. In our experiments, the convolutional based residual block always has a skip connection. However, for the LSTM layers, we did not find skip connections necessary as we also found in Chapter 5. All of the layers use the identity shortcut, and we did not find projection shortcuts to be helpful.

6.3 Experiments

We experimented with the Wall Street Journal (WSJ) ASR task. We used the standard configuration si284 dataset for training, dev93 for validation and eval92 for test

evaluation sets. Our input features were 80 dimensional filterbanks computed every 10ms with delta and delta-delta differences normalized by per speaker mean and variance. The baseline EncodeRNN function is a 3 layer BLSTM with 256 LSTM units per-direction (or 512 total) and $4 = 2^2$ time factor reduction. The DecodeRNN is a 1 layer LSTM with 256 LSTM units. All the weight matrices were initialized with a uniform distribution $\mathcal{U}(-0.1, 0.1)$ and bias vectors of 0. For the convolutional model, all the filter matrices were initialized with a truncated normal distribution $\mathcal{N}(0, 0.1)$, and used 32 output channels. Gradient norm clipping to 1 was applied, together with Gaussian weight noise $\mathcal{N}(0, 0.075)$ and L2 weight decay 1e$-$5 [31]. We used ADAM with the default hyperparameters described in [61]; however, we decayed the learning rate from 1e$-$3 to 1e$-$4 after it converged. We used 10 GPU workers for asynchronous SGD under the TensorFlow framework [2]. We monitored the dev93 Word Error Rate (WER) until convergence, and report the corresponding eval92 WER. The models took $O(5)$ days to converge.

6.3.1 Acronyms for different types of layers

All the residual blocks follow the structure of Fig. 6-3. Here are the acronyms for each component we use in the following subsections:

P / 2 subsampling projection layer.

C (f $\times$ t) convolutional layer with filter f and t under the frequency and time axes.

B batch normalization or (BatchNorm)

L bidirectional LSTM layer.

ResCNN residual block with convolutional layer inside.

ResConvLSTM residual block with convolutional LSTM layer inside.

6.3.2 Network in Network for Hierarchical Connections

We first begin by investigating the acoustic encoder depth of the baseline model without using any convolutional layers. Our baseline model follows [6] using the skip connection technique in its time reduction. The baseline L $\times$ 3, or 3 layer BLSTM acoustic encoder, model achieves a 14.76% WER.

When we simply increase the acoustic model encoder depth (i.e., to depth 8), the model does not converge well and we suspect the network to be trapped in a poor local minima. By using the projection subsampling layer as discussed in Section 6.2.3, we improve our WER to 13.61%, or a 7.8% relative gain over the baseline.

We can further increase the depth of the network by adding more NiN 1×1 convolution modules in between each LSTM layer. This improves our model's performance further to 12.88% WER, or a 12.7% relative improvement over the baseline. The BatchNorm layers were critical, and without them, we found the model did not converge well. Table 6.1 summarizes the results of applying network-in-network modules in the hierarchical subsampling process.

Table 6.1: We build deeper encoder networks by adding NiN modules in between LSTM layers.

Model	**WER**
L $\times$ 3	14.76
L $\times$ 8	Diverged
(L + P / 2 + B + R) $\times$ 2 + L	13.61
(L + P / 2 + B + R + C(1$\times$1) + BatchNorm + R) $\times$ 2 + L	12.88

6.3.3 Going Deeper with Convolutions and Residual Connections

In this subsection, we extend Section 6.3.2 and describe experiments in which we build deeper encoders by stacking convolutional layers and residual blocks in the acoustic encoder before the BLSTM. Unlike computer vision applications or truncated BPTT training in ASR, seq2seq models need to handle very long utterances (i.e., >2000

frames). If we simply stack a CNN before the BLSTMs, we quickly run out of GPU memory for deep models and also have excessive computation times. Our strategy to alleviate this problem is to apply striding in the first and second layer of the CNNs to reduce the time dimensionality and memory footprint.

We found no gains by simply stacking additional ResLSTM blocks even up to 8 layers. However, we do find gains if we use convolutions. If we stack 2 additional layers of 3×3 convolutions, our model improves to 11.80% WER, or a 20% relative over the baseline. If we take this model and add 8 residual blocks (for a total of $(2 + (8)2 + 5) = 23$ layers in the encoder), our model further improves to 11.11% WER, or a 24.7% relative improvement over the baseline. We found that using 8 residual blocks slightly outperforms 4 residual blocks. Table 6.2 summarizes the results of these experiments.

Table 6.2: We build deeper encoder networks by adding convolution and residual network blocks. The NiN block equals (L + C (1x1) + B + R) × 2 + L).

Model	**WER**
L × 3	14.76
NiN (from Section 6.3.2)	12.88
ResLSTM × 8	15.00
(C (3 × 3) / 2) × 2 + NiN	11.80
(C (3 × 3) / 2) × 2 + ResCNN × 4 + NiN	11.30
(C (3 × 3) / 2) × 2 + ResCNN × 8 + NiN	11.11

6.3.4 Convolutional LSTM Experiments

In this subsection, we investigate the effectiveness of the convolutional LSTM. Table 6.3 compares the effect of using convolutional LSTM layers. It can be observed that a pure ConvLSTM performs much worse than the baseline — we still need the fully connected LSTM[1]. However, replacing the ResConv block with ResConvLSTM as shown in Figure 6-4 gives us an additional 7% relative gains. In our experiments, we

[1]We only use 32 output channels. Thus it can be improved if we increase the channel size.

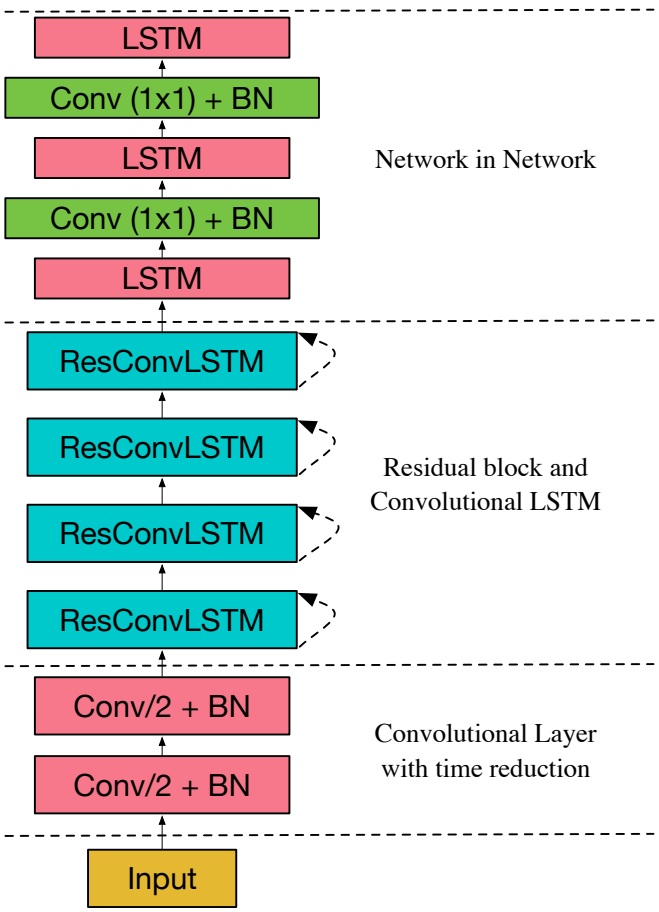


Figure 6-4: Our best model: includes two convolutional layers at the bottom, followed by four residual blocks and LSTM NiN blocks. Each residual block contains one convolutional LSTM layer and one convolutional layer.

always use 3×1 filters for ConvLSTM because the recurrent structure captures temporal information, while the convolutions capture spectral structure. We conjecture that the gain is because the convolutional recurrent state maintains spectral structure and reduces overfitting.

Table 6.4 compares our WSJ results with other published end-to-end models. To our knowledge, the previous best reported WER on WSJ without an LM was the seq2seq model with Task Loss Estimation achieving 18.0% WER in [7]. Our baseline, also a seq2seq model, achieved 14.76% WER. Our model is different from that of [7] in that we did not use location-based priors on the attention model and we used weight noise. Our best model, shown in Figure 6-4, achieves a WER of 10.53%.

Table 6.3: Performance of models with convolutional LSTM layers. The NiN block equals (L + C (1x1) + B + R) × 2 + L).

Model	**WER**
L × 3	14.76
ConvLSTM × 3	24.23
(C (3×3)) × 2 + ResCNN × 4 + NiN	11.30
(C (3×3)) × 2 + ResConvLSTM (3 × 1) × 4 + NiN	10.53

Table 6.4: Wall Street Journal test eval92 Word Error Rate (WER) results across Connectionist Temporal Classification (CTC) and Sequence-to-sequence (seq2seq) models. The models were decoded without a dictionary or language model.

Model	**WER**
CTC (Graves et al., 2014) [34]	30.1
seq2seq (Bahdanau et al., 2016) [7]	18.0
seq2seq + deep convolutional (our work)	10.53

6.4 Summary

In this chapter, we explored very deep CNNs for end-to-end speech recognition. We applied Network-in-Network principles to add depth and non-linearities to hierarchical RNNs. We also applied Batch Normalization and Residual connections to build very deep convolutional towers to process the acoustic features. Finally, we also explored Convolutional LSTMs, wherein we replaced the inner product of LSTMs with convolutions to maintain spectral structure in its representation. Together, we added more expressive capacity to build a very deep model without substantially increasing the number of parameters. On the WSJ ASR task, we obtained 10.5% WER without a language model, an 8.5% absolute improvement over the previously published best result [6]. Compared to chapter 5, the seq2seq model gains more from a very deep structure. We believe this is because it has more constraints for the representation power in a hybrid framework. This also the suggests seq2seq framework has more potential to incorporate more information to bring down the word error rate further.

7 Conclusions

7.1 Summary

In this thesis, I have explored the use of deep learning models for acoustic modeling in automatic speech recognition. I have made contributions in four major areas.

1. I present several new algorithms to speed-up recurrent neural network training. By combining these algorithms with a well-engineered deep learning framework, we present the first flexible and large-scale system that allows researchers to explore different neural architectures for automatic speech recognition. In particular, we propose an efficient batching algorithm to speed-up unidirectional RNN training. We further introduce latency-controlled bidirectional LSTMs (BLSTMs) which can exploit the entire input history while keeping latency under control. Moreover, we present a two-forward-pass procedure to speed up sequence-discriminative training when memory is the main constraint.

2. In the low-resource scenario, I propose a prediction-adaptation-correction RNN (PAC-RNN). A PAC-RNN is comprised of a pair of neural networks in which a *correction* network uses the auxiliary information given by a *prediction* network to help estimate the state probability. The information from the correction network is also used by the prediction network in a recurrent loop. Our model outperforms other state-of-the-art neural networks (DNNs, LSTMs) on several low-resource languages. Moreover, transfer learning from a language that is similar to the target language can help improve performance further.

3. I extend deep long short-term memory (DLSTM) recurrent neural networks by introducing gated direct connections between memory cells in adjacent layers. These direct links, called highway connections, enable unimpeded information flow across different layers and thus alleviate the gradient vanishing problem when building deeper LSTMs. We further introduce gridLSTMs which can exploit highway information more effectively. Experiments on different benchmark datasets such as AMI, SWBD, HKUST, GALE Mandarin and Arabic MGB indicate that we can train deeper LSTMs and achieve better improvement from sequence training with highway LSTMs (HLSTMs). Our novel model obtains a 5% to 10% relative WER reduction on eval sets, outperforming state-of-the-art NN models on various datasets.

4. I successfully train very deep convolutional networks to add more expressive power and better generalization for end-to-end ASR models. I apply network-in-network principles, batch normalization, residual connections, and convolutional LSTMs to build very deep recurrent and convolutional structures. Our models exploit the spectral structure in the feature space and add computational depth without overfitting issues. I experiment with the WSJ ASR benchmark task and achieve 10.5% word error rate without any dictionary or language model using a 15 layer deep network, significantly outperforming state-of-the-art seq2seq models.

7.2 Future Work and Directions

Several future research opportunities arise from the work presented in this thesis. We briefly discuss a few of them below.

- **Prediction framework using more auxiliary information** In this thesis, we only used information about the next phoneme in the PAC-RNN model. We can further extend this prediction framework to utilize additional auxiliary information such as speaking rate, duration, and word/syllable boundary.

Predict the speaking rate The inherent speaking rate often varies across talkers, and utterances. Evidence has already been shown that ASR performance can be improved via data augmentation by perturbing the speaking rate [76]. We believe that the PAC-RNN is a natural way to estimate speaking rate in the prediction model, and can potentially use this information in the primary acoustic model. Moreover, traditionally, we assume that the speaking rate is fixed when we combine the acoustic score and language model score. However, the interpolation weights should change based on the speaking rate. Therefore, to combine the language model score based the predicted speaking rate may also be a promising research direction to explore.

Predict the duration of acoustic units All research in this thesis was based on frame-level features. It is well-known that this feature has limitation for duration modeling. To eliminate these limitations, many techniques have been developed. These techniques can be described in a unified framework named the segmental model [75]. We believe that by predicting duration information for the raw input as a binary mask, that segment-level features can be more easily incorporated into neural network models.

Predict the word/syllable boundaries In all our experiments, we used a fixed computational network graph. The best neural network structure may not be static and may be a function of the input sequence. We believe the predicted boundary information could help us control the unrolling behavior of the network. For example, a predicted word boundary can force the acoustic model to only learn the long-term dependency from the speech signal instead of text by adding a forget gate, which we can reset when we become aware of a new word boundary.

- **Optimize the structure to reduce model parameters** For good quality recording conditions in English, ASR performance has already attained close to human parity. However, it usually requires a large amount of computation

which is not feasible in an online system. It is an interesting research topic to optimize the model structure, to reduce the computation and latency. Evidence has already been observed in [68] which proposes a simpler version of LSTM but achieves similar performance for language modeling.

- **More Data for seq2seq model** Our seq2seq model has shown state-of-the-art results on WSJ. However, the performance gap between our end-to-end model and HMM based systems is still more than 2%. One hypothesis is that our model is simply too powerful, and too easy to overfit. We believe our model will perform better on larger datasets. Moreover, since the seq2seq model can optimize the LM and AM jointly, we should leverage on the virtually unlimited amount of acoustic-only data, and text-only data that we have. Possible methods include pre-training with a generative model for both encoder and decoders. Evidence has already been shown in machine translation that this may be a promising research direction [116, 43].

Glossary of Acronyms

AM Acoustic Model. 3

AMI A English meeting recording corpus. 39

ASR Automatic Speech Recognition. 3

BatchNorm Batch Normalization. 97

BLSTM Bidirectional Long Short-Term Memory. 48

BN Bottleneck. 31

BN-CMLLR BN features with Constrained Maximum Likelihood Linear Regression. 71

BPTT Back-Propagation Through Time. 36

CD Context Dependent. 28

CE Cross Entropy. 29

CI Context Independent. 28

CMLLR Constrained Maximum Likelihood Linear Regression. 68

CN Computational Network. 44

CNN Convolutional Neural Network. 35

CNTK Computational Network ToolKit. 22

ConvLSTM Convolutional LSTM. 97

CSC-BPTT Context-Sensitive-Chunk BPTT. 53

DAG Directed Acyclic Graph. 46

DLSTM Deep Long Short-Term Memory. 37

DNN Deep Neural Network. 3

FBANK Mel-Frequency Log-Filterbank. 64

FLP Full Language Pack. 39

fMLLR Feature-Space Maximum Likelihood Linear Regression. 68

GALE A Chinese broadcast conversation speech corpus. 40

GLSTM Grid LSTM. 83

GMM Gaussian Mixture Model. 21

GPU Graphical Processing Units. 30

HKUST A conversational telephone speech corpus from Mandarin speakers. 40

HLSTM Highway LSTM. 81

HMM Hidden Markov Model. 26

IARPA-Babel A multilingual corpus. 38

LAS Listen, Attend and Spell. 99

LC-BLSTM Latency-Controlled BLSTM. 53

LID Language Identification. 72

LM Language Model. 26

LrSBN Low-Rank Stacked Bottleneck. 34

LSTM Long Short-Term Memory. 21

LSTMP Long Short-Term Memory with a Projection layer. 38

LVCSR Large-Vocabulary Continuous Speech Recognition. 37

MFCC Mel-frequency Cepstral Coecients. 25

MGB A modern arabic corpus used in this thesis. 40

ML Maximum Likelihood. 28

NiN Network-in-Network. 96

NN Neural Network. 21

npGLSTM non-Prioritized GLSTM. 85

OOV Out-of-Vocabulary. 74

PAC-RNN Prediction-Adaptation-Correction Recurrent Neural Network. 61

pGLSTM Prioritized GLSTM. 85

PLP Perceptual Linear Prediction. 25

ReLU Rectified Linear Unit. 102

ResNets Residual Network. 97

RLSTM Residual LSTM. 83

RNN Recurrent Neural Network. 35

SBN Stacked Bottleneck. 33

SCC Strongly Connected Component. 49

seq2seq Sequence-to-Sequence. 96

SGD Stochastic Gradient Descent. 31

sMBR state-level Minimum Bayes Risk. 29

SWBD A English conversational based corpus. 39

TIMIT A phone recognition corpus. 38

WER Word Error Rate. 3

WFST Weighted Finite State Transducer. 27

Bibliography

[1] *IARPA broad agency announcement IARPA-BAA-11-02*, 2011.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[3] Ossama Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2013.

[4] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in English and Mandarin. In *Proc. International Conference on Machine Learning (ICML)*, 2016.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine

translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[6] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[7] Dzmitry Bahdanau, Dmitriy Serdyuk, Philemon Brakel, Nan Rosemary Ke, Jan Chorowski, Aaron Courville, and Yoshua Bengio. Task loss estimation for sequence prediction. In *International Conference on Learning Representations Workshop*, 2016.

[8] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[9] Jean Carletta. Unleashing the killer corpus: experiences in creating the multi-everything ami meeting corpus. *Language Resources and Evaluation*, 41(2): 181–190, 2007.

[10] William Chan and Ian Lane. Deep convolutional neural networks for acoustic modeling in low resource languages. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[11] William Chan and Ian Lane. On online attention-based speech recognition and joint Mandarin character-pinyin training. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2016.

[12] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[13] Kai Chen and Qiang Huo. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5880–5884. IEEE, 2016.

[14] Kai Chen, Zhi-Jie Yan, and Qiang Huo. Training deep bidirectional LSTM acoustic model for LVCSR by a context-sensitive-chunk bptt approach. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2015.

[15] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[16] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Proc. Neural Information Processing Systems (NIPS)*, 2015.

[17] Ekapol Chuangsuwanich. *Multilingual techniques for low resource automatic speech recognition*. PhD thesis, Massachusetts Institute of Technology, 2016.

[18] Ekapol Chuangsuwanich, Yu Zhang, and James Glass. Multilingual data selection for training stacked bottleneck features. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[19] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[20] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[21] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20:30–42, January 2012.

[22] Namrata Dave. Feature extraction methods LPC, PLP and MFCC in speech recognition. *International Journal For Advance Research in Engineering And Technology, ISSN 2320-6802*, 1(VI):1–5, 2013.

[23] Li Deng and Jianshu Chen. Sequence classification using the high-level features extracted from deep neural networks. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6844–6848. IEEE, 2014.

[24] Jonathan G Fiscus, Jerome Ajot, Nicolas Radde, and Christophe Laprun. Multiple dimension levenshtein edit distance calculations for evaluating automatic speech recognition systems during simultaneous speech. In *The International Conference on language Resources and Evaluation (LERC)*, 2006.

[25] Mark JF Gales. Semi-tied covariance matrices for hidden markov models. *IEEE transactions on speech and audio processing*, 7(3):272–281, 1999.

[26] Mark JF Gales, Kate M Knill, and Anton Ragni. Unicode-based graphemic systems for limited resource languages. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5186–5190. IEEE, 2015.

[27] John S Garofolo, Lori F Lamel, William M Fisher, Jonathon G Fiscus, and David S Pallett. Darpa TIMIT acoustic-phonetic continous speech corpus cd-rom. NIST speech disc 1-1.1. *NASA STI/Recon technical report n*, 93, 1993.

[28] Pegah Ghahremani, Jasha Droppo, and Michael L. Seltzer. Linearly augmented deep neural network. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[29] James R Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech & Language*, 17(2):137–152, 2003.

[30] John J Godfrey, Edward C Holliman, and Jane McDaniel. Switchboard: Telephone speech corpus for research and development. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 517–520. IEEE, 1992.

[31] Alex Graves. Practical variational inference for neural networks. In *Proc. Neural Information Processing Systems (NIPS)*, 2011.

[32] Alex Graves. Sequence transduction with recurrent neural networks. In *International Conference on Machine Learning: Representation Learning Workshop*, 2012.

[33] Alex Graves. Generating sequences with recurrent neural networks. In *arXiv:1308.0850*, 2013.

[34] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proc. International Conference on Machine Learning (ICML)*, 2014.

[35] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with bidirectional LSTM. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2013.

[36] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[37] František Grézl and Martin Karafiát. Hierarchical neural net architectures for feature extraction in ASR. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2010.

[38] Frantisek Grézl and Martin Karafiát. Adapting multilingual neural network hierarchy to a new language. In *Spoken Language Technologies for Under-Resourced Languages*, 2014.

[39] Frantisek Grézl, Martin Karafiát, Stanislav Kontár, and Jan Cernocky. Probabilistic and bottle-neck features for LVCSR of meetings. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 4, pages IV–757. IEEE, 2007.

[40] František Grézl, Martin Karafiat, and Miloš Janda. Study of probabilistic and bottle-neck features in multilingual environment. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 359–364, 2011.

[41] František Grézl, Ekaterina Egorova, and Martin Karafiát. Further investigation into multilingual training and adaptation of stacked bottle-neck neural network structure. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 48–53. IEEE, 2014.

[42] Frantisek Grézl, Martin Karafiát, and Karel Vesely. Adaptation of multilingual stacked bottle-neck neural network structure for new language. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7654–7658. IEEE, 2014.

[43] Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loic Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. On using monolingual corpora in neural machine translation. In *arXiv:1503.03535*, 2015.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[45] Georg Heigold, Erik McDermott, Vincent Vanhoucke, Andrew Senior, and Michiel Bacchiani. Asynchronous stochastic optimization for sequence training of deep neural networks. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5587–5591. IEEE, 2014.

[46] Salah Hihi and Yoshua Bengio. Hierarchical Recurrent Neural Networks for Long-Term Dependencies. In *Proc. Neural Information Processing Systems (NIPS)*, 1996.

[47] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, November 2012.

[48] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. In *arXiv*, 2012.

[49] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[50] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[51] Wei-Ning Hsu, Yu Zhang, and James Glass. A prioritized grid long short-term memory rnn for speech recognition. In *Spoken Language Technology Workshop (SLT), 2016 IEEE*, pages 467–473. IEEE, 2016.

[52] Wei-Ning Hsu, Yu Zhang, Ann Lee, and James Glass. Exploiting depth and highway connections inconvolutional recurrent deep neural networks for speech recognition. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, pages 395–399, 2016.

[53] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7304–7308. IEEE, 2013.

[54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. International Conference on Machine Learning (ICML)*, 2015.

[55] Navdeep Jaitly, Vincent Vanhoucke, and Geoffrey Hinton. Autoregressive product of multi-frame predictions can improve the accuracy of hybrid models. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[56] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[57] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. In *International Conference on Learning Representations*, 2016.

[58] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[59] Martin Karafiát, Frantisek Grézl, Mirko Hannemann, Karel Veselỳ, and Jan Cernockỳ. BUT babel system for spontaneous cantonese. In *Interspeech*, volume 13, pages 2589–2593, 2013.

[60] Martin Karafiát, Karel Veselỳ, Igor Szoke, Lukáš Burget, František Grézl, Mirko Hannemann, and Jan Černockỳ. BUT ASR system for babel surprise evaluation 2014. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, pages 501–506. IEEE, 2014.

[61] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[62] Brian Kingsbury, Tara Sainath, and Hagen Soltau. Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free

optimization. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2012.

[63] KM Knill, Mark JF Gales, Shakti P Rath, Philip C Woodland, Chao Zhang, and S-X Zhang. Investigation of multilingual deep neural networks for spoken term detection. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 138–143. IEEE, 2013.

[64] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. Neural Information Processing Systems (NIPS)*, 2012.

[65] Viet-Bac Le, Lori Lamel, Abdel Messaoudi, William Hartmann, Jean-Luc Gauvain, Cécile Woehrling, Julien Despres, and Anindya Roy. Developing STT and KWS systems using limited language resources. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[66] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86: 2278–2324, November 1998.

[67] Byung Suk Lee. *Noise robust pitch tracking by subband autocorrelation classification.* Columbia University, 2012.

[68] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. *Proc. International Conference on Machine Learning (ICML)*, 2017.

[69] Xiangang Li and Xihong Wu. Improving long short-term memory networks using maxout units for large vocabulary speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4600–4604. IEEE, 2015.

[70] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *International Conference on Learning Representations*, 2013.

[71] Christopher Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[72] Yajie Miao, Hao Zhang, and Florian Metze. Distributed learning of multilingual dnn feature extractors using GPUs. *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[73] Tomas Mikolov, Karafiat Martin, Burget Luka, Eernocky Jan, and Khudanpur Sanjeev. Recurrent neural network based language model. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2010.

[74] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[75] Mari Ostendorf, Vassilios V Digalakis, and Owen A Kimball. From HMM's to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on speech and audio processing*, 4(5):360–378, 1996.

[76] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Proceedings of INTERSPEECH*, pages 2440–2444. ISCA, 2015.

[77] Daniel Povey. Discriminative training for large vocabulary speech recognition. In *PhD Dissertation, Cambridge University Engineering Department*, 2003.

[78] Daniel Povey and Kaisheng Yao. A basis method for robust estimation of constrained MLLR. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 4460–4463. IEEE, 2011.

[79] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannenmann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi speech recognition toolkit. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.

[80] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of DNNs with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*, 2014.

[81] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, pages 2751–2755, 2016.

[82] Kristin Precoda. Non-mainstream languages and speech recognition: Some challenges. *CALICO journal*, pages 229–243, 2004.

[83] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, February 1989.

[84] Lawrence R Rabiner and Biing-Hwang Juang. Fundamentals of speech recognition. *PTR Prentice Hall*, 1993.

[85] Tara Sainath, Brian Kingsbury, Abdel-rahman Mohamed, George E. Dahl, George Saon, Hagen Soltau, Tomas Beran, Aleksandr Y. Aravkin, and Bhuvana Ramabhadran. Improvements to deep convolutional neural networks for

LVCSR. In *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2013.

[86] Tara Sainath, Brian Kingsbury, Abdel-rahman Mohamed, and Bhuvana Ramabhadran. Learning filter banks within a deep neural network framework. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 297–302. IEEE, 2013.

[87] Tara Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6655–6659. IEEE, 2013.

[88] Tara Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[89] Tara Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[90] Hasim Sak, Andrew Senior, and Francoise Françoise. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[91] Hasim Sak, Oriol Vinyals, Georg Heigold, Andrew Senior, Erik McDermott, Rajat Monga, and Mark Mao. Sequence discriminative distributed training of long short-term memory recurrent neural networks. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[92] Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 24–29. IEEE, 2011.

[93] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2011.

[94] Michael L Seltzer, Dong Yu, and Yongqiang Wang. An investigation of deep neural networks for noise robust speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7398–7402. IEEE, 2013.

[95] Tom Sercu and Vaibhava Goel. Advances in very deep convolutional neural networks for LVCSR. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2016.

[96] Tom Sercu, Christian Puhrsch, Brian Kingsbury, and Yann LeCun. Very deep multilingual convolutional neural networks for LVCSR. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[97] Xingjian Shi, Zhourong Chen, Hao Wang, Dit Yan Yeung, Wai Kin Wong, and Wang Chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Proc. Neural Information Processing Systems (NIPS)*, 2015.

[98] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. International Conference on Learning Representation (ICLR)*, 2015.

[99] Ediz Sohoglu and Matthew H Davis. Perceptual learning of degraded speech by minimizing prediction error. *Proceedings of the National Academy of Sciences*, 113(12):E1747–E1756, 2016.

[100] Ediz Sohoglu, Jonathan E Peelle, Robert P Carlyon, and Matthew H Davis. Predictive top-down integration of prior knowledge during speech perception. *Journal of Neuroscience*, 32(25):8443–8453, 2012.

[101] Frank K Soong and E-F Huang. A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 705–708. IEEE, 1991.

[102] Rupesh Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *International Conference on Machine Learning: Deep Learning Workshop*, 2015.

[103] Ilya Sutskever, Oriol Vinyals, and Quoc Le. Sequence to sequence learning with neural networks. In *Proc. Neural Information Processing Systems (NIPS)*, 2014.

[104] Pawel Swietojanski, Arnab Ghoshal, and Steve Renals. Convolutional neural networks for distant speech recognition. *IEEE Signal Processing Letters*, 21(9): 1120–1124, 2014.

[105] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[106] Tian Tan, Yanmin Qian, Dong Yu, Souvik Kundu, Liang Lu, Khe Chai Sim, Xiong Xiao, and Yu Zhang. Speaker-aware training of LSTM-RNNs for acoustic

modelling. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5280–5284. IEEE, 2016.

[107] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[108] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, volume 5, 2015.

[109] László Tóth. Convolutional deep maxout networks for phone recognition. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[110] Zoltán Tüske, Pavel Golik, David Nolden, Ralf Schlüter, and Hermann Ney. Data augmentation, feature combination, and multilingual neural networks to improve ASR and KWS performance for low-resource languages. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[111] Fabio Valente, Jithendra Vepa, Christian Plahl, Christian Gollan, Hynek Hermansky, and Ralf Schlüter. Hierarchical neural networks feature extraction for LVCSR system. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2007.

[112] Karel Veselỳ, Martin Karafiát, František Grézl, Miloš Janda, and Ekaterina Egorova. The language-independent bottleneck features. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, pages 336–341. IEEE, 2012.

[113] Karel Vesely, Arnab Ghoshal, Lukas Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2013.

[114] Chao Weng, Dong Yu, Shinji Watanabe, and Fred Jung. Recurrent deep neural networks for robust speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[115] Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4): 490–501, 1990.

[116] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey

Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. In *arXiv:1609.08144*, 2016.

[117] Kaisheng Yao and Geoffrey Zweig. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. In *INTERSPEECH*, 2015.

[118] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated LSTM. Technical report, Microsoft Research, 2015. URL `http://arxiv.org/abs/1508.03790`.

[119] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112*, 2014.

[120] Le Zhang, Damianos Karakos, William Hartmann, Roger Hsiao, Richard Schwartz, and Stavros Tsakalidis. Enhancing low resource keyword spotting with automatically retrieved web documents. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2015.

[121] Yu Zhang, Ekapol Chuangsuwanich, and James Glass. Extracting deep neural network bottleneck features using low-rank matrix factorization. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 185–189. IEEE, 2014.

[122] Yu Zhang, Ekapol Chuangsuwanich, and James Glass. Language ID-based training of multilingual stacked bottleneck features. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[123] Yu Zhang, Dong Yu, Michael L Seltzer, and Jasha Droppo. Speech recognition with prediction-adaptation-correction recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5004–5008. IEEE, 2015.

[124] Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yao, Sanjeev Khudanpur, and James Glass. Highway long short-term memory RNNs for distant speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[125] Yu Zhang, Ekapol Chuangsuwanich, James Glass, and Dong Yu. Prediction-adaptation-correction recurrent neural networks for low-resource language speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5415–5419. IEEE, 2016.

[126] Yu Zhang, William Chan, and Navdeep Jaitly. Very deep convolutional networks for end-to-end speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4845–4849. IEEE, 2017.