This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author. The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author. When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.
Recurrent Neural Network Language Models for Automatic Speech Recognition

Siva Reddy Gangireddy

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2017
Abstract

The goal of this thesis is to advance the use of recurrent neural network language models (RNNLMs) for large vocabulary continuous speech recognition (LVCSR). RNNLMs are currently state-of-the-art and shown to consistently reduce the word error rates (WERs) of LVCSR tasks when compared to other language models. In this thesis we propose various advances to RNNLMs. The advances are: improved learning procedures for RNNLMs, enhancing the context, and adaptation of RNNLMs. We learned better parameters by a novel pre-training approach and enhanced the context using prosody and syntactic features.

We present a pre-training method for RNNLMs, in which the output weights of a feed-forward neural network language model (NNLM) are shared with the RNNLM. This is accomplished by first fine-tuning the weights of the NNLM, which are then used to initialise the output weights of an RNNLM with the same number of hidden units. To investigate the effectiveness of the proposed pre-training method, we have carried out text-based experiments on the Penn Treebank Wall Street Journal data, and ASR experiments on the TED lectures data. Across the experiments, we observe small but significant improvements in perplexity (PPL) and ASR WER.

Next, we present unsupervised adaptation of RNNLMs. We adapted the RNNLMs to a target domain (topic or genre or television programme (show)) at test time using ASR transcripts from first pass recognition. We investigated two approaches to adapt the RNNLMs. In the first approach the forward propagating hidden activations are scaled - learning hidden unit contributions (LHUC). In the second approach we adapt all parameters of RNNLM. We evaluated the adapted RNNLMs by showing the WERs on multi genre broadcast speech data. We observe small (on an average 0.1% absolute) but significant improvements in WER compared to a strong unadapted RNNLM model.

Finally, we present the context-enhancement of RNNLMs using prosody and syntactic features. The prosody features were computed from the acoustics of the context words and the syntactic features were from the surface form of the words in the context. We trained the RNNLMs with word duration, pause duration, final phone duration, syllable duration, syllable F0, part-of-speech tag and Combinatory Categorial Grammar (CCG) supertag features. The proposed context-enhanced RNNLMs were evaluated by reporting PPL and WER on two speech recognition tasks, Switchboard and TED lectures. We observed substantial improvements in PPL (5% to 15% relative) and small but significant improvements in WER (0.1% to 0.5% absolute).
The goal of this thesis is to advance the use of recurrent neural network language models (RNNLMs) for large vocabulary continuous speech recognition (LVCSR). RNNLMs are currently state-of-the-art and shown to consistently reduce the word error rates (WERs) of LVCSR tasks when compared to other language models. In this thesis we propose various advances to RNNLMs. The advances are: improved learning procedures for RNNLMs, enhancing the context, and adaptation of RNNLMs. We learned better parameters by a novel pre-training approach and enhanced the context using prosody and syntactic features.

We present a pre-training method for RNNLMs, in which the output weights of a feed-forward neural network language model (NNLM) are shared with the RNNLM. This is accomplished by first fine-tuning the weights of the NNLM, which are then used to initialise the output weights of an RNNLM with the same number of hidden units. Next, we present unsupervised adaptation of RNNLMs. We adapted the RNNLMs to a target domain (topic or genre or show) at test time using ASR transcripts from first pass recognition. We investigated two approaches to adapt the RNNLMs. In the first approach the forward propagating hidden activations are scaled - learning hidden unit contributions (LHUC). In the second approach we adapt all parameters of RNNLM. Finally, we present the context-enhancement of RNNLMs using prosody and syntactic features. The prosody features were computed from the acoustics of the context words and the syntactic features were from the surface form of the words in the context. We trained the RNNLMs with word duration, pause duration, final phone duration, syllable duration, syllable F0, part-of-speech tag and Combinatory Categorial Grammar (CCG) supertag features. We evaluated the proposed RNNLMs by showing the perplexity and WERs on TED lecture transcription task, Switchboard corpus and multi genre broadcast speech corpus.
Acknowledgements

First, I would like to express my gratitude to my supervisor Steve Renals for his supervision, encouragement, support and for giving me the opportunity to pursue a PhD at The Centre for Speech Technology Research (CSTR). I am also thankful to him for giving me freedom to explore new ideas all the time.

Many thanks to my secondary supervisors, Fergus McInnes and Junichi Yamagishi for valuable discussions and suggestions in the first year of my PhD. I am also grateful to Hiroshi Shimodaria and Simon King for serving in annual review internal committees.

I am extremely grateful to Mikko Kurimo and Simon King for examining this thesis and giving me valuable feedback.

Many thanks to everybody at CSTR for giving me feedback during annual reviews and networking over lunch breaks, coffee breaks and Friday evening pub. Especially Peter Bell and Pawel Swietojanski for many discussions on technical details of language modelling experiments.

I am grateful to Peter Bell and Catherine Lai for proof-reading part of this thesis.

I would like to thank my Indian friends Siva Reddy, Bharat, Spandana, Praveen, Srikanth and Rupa for making my time in Edinburgh something unique in my life.

I am extremely thankful to Akinobu Lee, Yohihiko Nankaku and Keichii Tokuda for supporting and supervising my internship at Nagoya Institute of Technology, Nagoya, Japan.

Many thanks to Apple Siri for giving me an internship to work on a real-time project. Also thanks to Murat Akbacak and Xiaochuan Niu for supervising the internship.

Many thanks to Level3 staff and Caroline Hastings for administrative help and organising my conference travel.

Finally I would like thank my parents and wife (Anusha) for their support and motivation all these years.

This work is supported by the Core Research for Evolitional Science and Technology (CREST) from the Japan Science and Technology Agency (JST) (uDialogue project).
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Siva Reddy Gangireddy)
Table of Contents

1 Introduction .................................................. 1
   1.1 Motivation ............................................. 1
   1.2 Contributions ........................................ 3
   1.3 Thesis Outline ....................................... 4
   1.4 Publications .......................................... 5

2 Statistical Language Models ............................. 7
   2.1 Automatic Speech Recognition ..................... 7
      2.1.1 Feature Extraction .............................. 8
      2.1.2 Acoustic Model .................................. 9
      2.1.3 Lexicon ........................................... 9
      2.1.4 Language Model .................................. 9
      2.1.5 Decoder ......................................... 10
   2.2 Language Modelling .................................. 10
      2.2.1 Evaluation Metric ............................... 11
   2.3 N–grams ............................................... 12
      2.3.1 Smoothing ....................................... 12
   2.4 Beyond N–grams ..................................... 17
      2.4.1 Class Language Models ......................... 18
      2.4.2 Cache Language Models ......................... 18
      2.4.3 Maximum Entropy Models ....................... 19
   2.5 Neural Network Language Models ................... 20
      2.5.1 Feed-forward Neural Network Language Models . 21
      2.5.2 Recurrent Neural Network Language Models . 24
      2.5.3 Long-Short Term Memory Recurrent Neural Networks . 28
      2.5.4 Gated Recurrent Neural Networks ............... 30
      2.5.5 Recurrent Neural Networks with Attention .... 30
## List of Figures

2.1 Block diagram of automatic speech recognition

2.2 Feed-forward neural network language model [Bengio et al., 2003; Schwenk, 2007]

2.3 Recurrent neural network language model [Mikolov et al., 2010, 2011a]

2.4 Back propagation through time (BPTT) [Werbos, 1990]

2.5 LSTM neuron with memory cell and gating functions

3.1 N–best list rescoring setup

4.1 Feed-forward neural network language model. Here the output layer weights are shared with output layer weights of the RNNLM (shown in red)

4.2 Recurrent neural network language model. Here the output layer weights are initialised with the output layer weights of NNLM

4.3 Perplexities of 3-gram, NNLM, RNNLM and PT-RNNLM on validation data of PTB. The blue and green bars represents the PPL before and after interpolation, respectively.

4.4 Perplexities of 3-gram, NNLM, RNNLM and PT-RNNLM on test data of PTB. The blue and green bars represents the PPL before and after interpolation, respectively.

4.5 The variation of PPL with respect to the number of pre-trained iterations (on test data). The same configuration of RNNLM and PT-RNNLM is used as described in Section 4.4.1

4.6 The variation of PPL with respect to the number of pre-trained iterations (on validation data). The same configuration of RNNLM and PT-RNNLM is used as described in Section 4.4.1
4.7 The number of iterations the RNNLM and PT-RNNLM took to converge to a local minimum. The same learning rate scheduler was used for both RNNLM and PT-RNNLM.

4.8 WERs of dev2010, tst2010 and tst2011 using Tandem MLAN HMM system. (a) Here the models were trained on 5M tokens of OOD and 2.4M tokens of in-domain data. (b) WERs of models which were trained on a total 12.4M tokens. (c) A total of 22.4M tokens were used to train the models, out of which 20M tokens were OOD data.

4.9 WERs of dev2010, tst2010 and tst2011 using Hybrid MLAN HMM system. (a) Here the models were trained on 5M tokens of OOD and 2.4M tokens of in-domain data. (b) WERs of models which were trained on a total 12.4M tokens. (c) A total of 22.4M tokens were used to train the models, out of which 20M tokens were OOD data.

5.1 Unsupervised adaptation framework

6.1 Recurrent neural network language model with a feature layer (green)

6.2 Alignments of an utterance at word level. These alignments are computed by force aligning the transcripts with the acoustic signal.

6.3 Word durations of an utterance computed by force aligning the transcripts with acoustic signal.

6.4 Pause durations of an utterance computed by force aligning the transcripts with acoustic signal.

6.5 Durations of the final phone preceding the pause.

6.6 (a) Perplexities on validation data, (b) Perplexities on Switchboard conversations of eval2000. The Prosody RNNLMs trained on word duration, pause duration, duration of final phone, syllable durations and syllable F0 features. Here the RNNLM-syldur and RNNLM-sylF0 models are trained on syllable context length of five. 3-gram LM is trained on combination of Switchboard and Fisher transcripts and the RNNLMs are trained only on part of Switchboard training transcripts (3.2M).

6.7 CCG derivation for demonstrating long-range dependency in a relative clause construction.
List of Tables

3.1 Best perplexities (PPL) reported so far on PTB test data in the literature 32
3.2 Number of tokens in train, valid and test set of PTB corpus 32
3.3 Number of tokens in various OOD data sources 33
3.4 Number of tokens in various OOD data sources before and after the data selection process 35
3.5 OOV rates of development and test data sets of TED task 35
4.1 Number of tokens after combining the in-domain data with subsets of OOD data. In-domain data consists of 2.4M tokens 50
5.1 % WERs of RNNLM and adapted RNNLM by LHUC method. The RNNLMs were trained on 640M tokens. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used 69
5.2 % WERs of RNNLM and adapted RNNLM by LHUC method. The RNNLMs are trained on 40M tokens. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used 69
5.3 % WERs of RNNLM and adapted RNNLM by adapting all parameters. During adaptation a learning rate of 0.1 is used. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used 70
5.4 % WERs of RNNLM and adapted RNNLM by adapting all the parameters. During adaptation a learning rate of 0.1 is used. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used 70
5.5 % WERs of RNNLM and adapted RNNLM by adapting subset of parameters. Adaptation learning rate is 0.1 71
5.6 % WERs of RNNLM and adapted RNNLM by adapting subset of parameters. Adaptation learning rate is 1.0. ...... 71

5.7 % WERs baseline RNNLM and adapted RNNLM by Adapt-all method. With a learning rate of 1.0 ...... 73

5.8 % WERs of RNNLM and RNNLM adapted on 10M tokens of acoustic transcripts. ...... 73

5.9 100-best oracle for dev.full. ...... 74

6.1 % WERs computed on 100-best lists of eval2000 data set (Switchboard conversations only). GMM acoustic model is used to force align the transcripts. ...... 87

6.2 % WERs computed on 100-best lists of eval2000 data set (Switchboard conversations only). GMM- and DNN-based acoustic models are used to force align the transcripts. ...... 88

6.3 PPLs and % WERs computed on tst2011 and 100-best lists of tst2011, respectively. DNN-HMM hybrid acoustic models trained on TED and AMI data are used to force align the transcripts and compute the prosody features. ...... 89

6.4 % WERs are computed on 100-best lists of eval2000 data set (Switchboard conversations only). Automatic syllable detection algorithm is used to get the boundaries of syllables. GMM-based acoustic models are used to get the word boundary information. ...... 90

6.5 % WERs are computed on 100-best lists of eval2000 data set (Switchboard conversations only). Before computing the F0 features the F0 sequence of each syllable is normalised using z-score normalisation. GMM-based acoustic models are used to get the word boundary information. ...... 91

6.6 POS and CCG Supertags of a sentence. ...... 95

6.7 Number of tokens after combining the in-domain data with subsets of OOD data. In-domain data consists of 2.4M tokens. ...... 96

6.8 The WERs of baseline 4-gram and RNNLMs trained on 7.4M and 12.4M tokens. ...... 97

6.9 The WERs of RNNLM trained on POS tags. Here the taggers are adapted to the language model training data. ...... 98
6.10 WERs of RNNLMs trained on 7.4M and 12.4M tokens and their POS tags ........................................ 98
6.11 WERs of dev2010 and tst2010 using models trained on CCG Supertags ....................................... 99
6.12 The utterances where POS tags helped to select a better hypothesis ............................................. 101
A.1 % WERs of dev.full at show level: before and after adaptation using LHUC and Adapt-all approaches  109
A.2 % WERs of dev.full at show level: before and after adaptation using LHUC and Adapt-all approaches  110
List of Symbols

$p_{abs}(w_i|w_{i-n+1}^{i-1})$ - Probability of a word $w_i$ given its context $w_{i-n+1}^{i-1}$ using absolute discount method.

c_t - Cell state of LSTM.

$c_i$ - Class assigned to a word $w_i$.

c(,) - Counts of occurrence of an n-gram.

$N_1(w_{i-n+1}^{i-1} \bullet)$ - Number of distinct words occurring after the context $w_{i-n+1}^{i-1}$.

$H(T)$ - Entropy of the test data.

e_o(t) - Errors at output layer.

$f_i$ - Output of forget gate of a LSTM.

$p_{GT}(w_i|w_{i-n+1}^{i-1})$ - Probability of a word $w_i$ given its context $w_{i-n+1}^{i-1}$ using Good-Turing discount method.

$h(t)$ - RNNLM hidden activations at time $t$.

$H$ - Hidden layer size.

$i_t$ - Output of input gate of a LSTM.

$p_{interp}(w_i|w_{i-n+1}^{i-1})$ - Probability of a word $w_i$ given its context $w_{i-n+1}^{i-1}$ using interpolation method.

$p_{katz}(w_i|w_{i-n+1}^{i-1})$ - Probability of a word $w_i$ given its context $w_{i-n+1}^{i-1}$ using katz back-off method.

$p_{KN}(w_i|w_{i-n+1}^{i-1})$ - Probability of a word $w_i$ given its context $w_{i-n+1}^{i-1}$ using Kneser-Ney smoothing method.
\( \alpha \) - Learning rate used to update the parameters of NNLM and RNNLM.

\( n_r \) - Number of \( n \)-grams that occur \( r \) times in the training data.

\( n_T \) - Total number of words in the test data.

\( o_t \) - Output of output gate of a LSTM.

\( P(W|A) \) - Posterior probability of sequence of words given the acoustic features.

\( P \) - Size of projection layer.

\( r_m \) - Show specific adaptation parameters.

\( T \) - Test data.

\( f(x_t : \theta) \) - Non-linear transfer function.

\( N \) - The number of words in the vocabulary.

\( N_{c_i} \) - Number of words in a class \( c_i \).

\( \hat{W} \) - Predicted sequence of words.

\( w_i \) - \( i^{th} \) word in the sequence of words.
Chapter 1

Introduction

This thesis is concerned with the language model (LM) within the automatic speech recognition (ASR) system. The LM is trained on text data and is used by the ASR system to generate most likely sequences of words that have generated the speech. Current state-of-the-art ASR systems use both \(n\)-grams and neural network based LMs for decoding and lattice rescoring. But neural network based LMs have shown significant improvements over standard \(n\)-grams [Bengio et al., 2003; Mikolov et al., 2010, 2011a; Mikolov, 2012; Sundermeyer et al., 2012].

Due to significant improvements and advantages over traditional \(n\)-grams this thesis investigates the neural network based LMs. The advantages are: 1) they provide inherent smoothing and generalise well to unseen sequences of words by utilising the similarities between the words (syntactic and semantic); 2) they are flexible making them easy to add features to and also easy to train and test with the features based on lexical and non-lexical information; 3) adaptation to a target domain is easier since these models are parametric; 4) they implicitly learn the long distance dependencies (using recurrent neural network language models).

1.1 Motivation

The famous visions of artificial intelligence include making machines communicate with humans in natural spoken language, play complex games (Chess and AlphaGo [Silver et al., 2016]) and most recently self-driving cars. Playing games and self-driving cars are out of scope of this thesis: here we look at how we can make machines communicate in natural language with humans. To solve this we observe the human language learning and understanding skills and use them to make the machine commu-
nicate in natural spoken language. The whole process of human language learning and understanding is very complex and it may not be possible to simulate the same with the machines.

But by using data and statistical algorithms we can enable machines to communicate in natural spoken language. To do this, first the speech needs to be recognised. Second, the recognised utterance needs to be processed to understand its semantics and to generate any response. Such a system thus consists of automatic speech recognition and natural language processing modules. A machine translation module is also required if the languages covered by the machine are different from the language used by the human to communicate. All these modules require a prediction model to predict sequences of words and automatic responses. A LM can do the job of prediction of sequences of words and automatic responses. For better prediction a LM needs to be trained on huge amounts of text data.

LMs are an integral part of ASR, machine translation and natural language understanding systems. Until recently, simple $n$–grams were state-of-art in language modelling. For the past few years, neural network based models have been extensively investigated for acoustic models [Dahl et al., 2012; Sak et al., 2014] and language models [Bengio et al., 2003; Mikolov, 2012; Cho et al., 2014; Sundermeyer et al., 2012], statistical machine translation [Cho et al., 2014; Wu et al., 2016; Sutskever et al., 2014] and for natural language processing [Collobert and Weston, 2008; Yao et al., 2014].

There are two types of neural network based language models: feed-forward neural network language models (NNLMs), and recurrent neural network language models (RNNLMs). NNLMs are similar to the standard $n$–grams, in that the length of the context is fixed, but the probability is estimated in a low-dimensional continuous space. RNNLMs have recurrent connections and these connections are responsible for learning long-distance dependencies. In this thesis we investigate RNNLMs in the context of ASR, since the RNNLMs learns the long distance dependencies and they are the current state-of-the-art for language modelling. The goal of thesis is to make RNNLMs that better predict sequences of words using new approaches and to automatically adapt RNNLMs to new target domains.
1.2 Contributions

The main contributions of this thesis are:

- Development of a new pre-training algorithm to learn the best parameters for recurrent neural network language models (RNNLMs)
  - We show that prediction accuracy of pre-trained RNNLMs is higher than RNNLMs
  - Pre-trained RNNLMs converge faster than the RNNLMs
  - Empirically we show that a few iterations of pre-training is sufficient to improve the prediction accuracy
  - Pre-training also useful for ASR in terms of WERs

- Adaptation of RNNLMs to a target domain (topic or genre or show) at test time
  - Investigated two adaptation methods 1) learning hidden unit contributions (LHUC) 2) adaptation by updating all the parameters of RNNLM
  - We showed that unsupervised adaptation of RNNLMs to target domain (BBC show) is useful for ASR in terms of WERs. The improvements are small but statistically significant
  - We also discussed various difficulties in adapting the RNNLMs to a target domain at test time

- Enhancement of the context of RNNLMs using prosody and syntactic features computed from the context of the current word
  - We showed that enhancing the context with prosody features improves the prediction accuracy
  - Out of all prosody features (word duration, pause duration final phone duration, syllable duration and syllable F0) duration features (word and syllable) outperform other features
  - We also showed that syntactic information of the words in the context improves the prediction accuracy
  - Both the prosody and syntactic features are useful for ASR in terms of WERs, except CCG supertags
1.3 Thesis Outline

Chapter 1 we present the automatic speech recognition (ASR) problem and review the literature of statistical language models, \( n \)-grams, and neural network based language models. The neural network based models include feed-forward neural network language models (NNLMs), recurrent neural network language models (RNNLMs) and long-short term memory (LSTMs) neural networks.

Chapter 2 consists of two parts. In the first part of the chapter we describe various datasets used to train and evaluate the proposed language models (LMs) for automatic speech recognition (ASR). In the second part of the chapter a description about experimental setup used to incorporate the proposed LMs into the ASR is given.

In Chapter 4, we propose a pre-training method for the recurrent neural network language model (RNNLM), by sharing the output weights of a feed forward neural network language model (NNLM) with the RNNLM. This is accomplished by first fine-tuning the weights of the NNLM, which are then used to initialise the output weights of an RNNLM with the same number of hidden units. We carried out text-based experiments on the Penn Treebank Wall Street Journal data, and ASR experiments on the TED talks data. Across the experiments, we observe small but significant improvements in perplexity and ASR word error rate.

In Chapter 5, we present the supervised and unsupervised discriminative adaptation of RNNLMs in a broadcast transcription task to target domains defined by either genre or show (show is a BBC programme. A typical BBC programme runs from 30 to 60 minutes). We investigated two approaches based on (1) scaling forward-propagated hidden activations (Learning Hidden Unit Contributions (LHUC) technique) and (2) adaptation of all the weights of RNNLM. To investigate the effectiveness of the proposed methods we carry out experiments on multi-genre broadcast (MGB) data following the MGB-2015 challenge protocol. We observe small but significant improvements in WER compared to a strong unadapted RNNLM model.

Chapter 6 presents the context enhancement of RNNLMs with prosodic and syntactic features computed using the context of the current word. First we enhanced the context with prosody features. Since it is plausible to compute the prosody features at the word and syllable level we have trained the models on prosody features computed at both these levels. To investigate the effectiveness of proposed models we report perplexity and WER for two speech recognition tasks, Switchboard and TED. We

\[ \text{http://www.mgb-challenge.org/english.html} \]
observed substantial improvements in perplexity and small improvements in WER. In addition to prosody features, we also enhanced the context with syntactic features computed from the context of the current word. The syntactic features are part-of-speech (POS) tags and Combinatory Categorial Grammar (CCG) supertags \cite{Steedman2000}. To investigate the effectiveness of these models we report the PPL and WERs on TED speech recognition task. We observed small but significant and consistent improvements across the datasets.

Finally we conclude and describe future work in Chapter 7.

1.4 Publications

Chapter 4, Chapter 5 and part of Chapter 6 have been published in international conferences. The work on feed-forward pre-training for RNNLMs is published in \cite{Gangireddy2014}. Chapter 5 covers the work on unsupervised adaptation of RNNLMs published in \cite{Gangireddy2016}. The part of Chapter 6 covering context enhancement of RNNLMs using prosody features is published in \cite{Gangireddy2015}.
Chapter 2

Statistical Language Models

In this chapter, we introduce the automatic speech recognition (ASR) problem and review the literature of statistical language models, n-grams, and neural network based language models. The neural network based models include feed-forward neural network language models (NNLMs), recurrent neural network language models (RNNLMs) and long-short term (LSTMs) memory neural networks.

2.1 Automatic Speech Recognition

Statistical language models are widely used in automatic speech recognition (ASR) and machine translation (MT), and are also used in other tasks like spelling correction and information retrieval. Given a speech signal, the task of ASR is to generate the sequence of words most likely to have generated the speech. A language model provides the prior probability of a sequence of words and constrains the search space for the best sequence of words during decoding. Given an acoustic signal \( A \), the sequence of words most likely to have generated \( A \) is determined by:

\[
\hat{W} = \arg \max_w [\log(P(W|A))] \tag{2.1}
\]

Since \( P(W|A) \) is difficult to compute directly, Bayes’ rule is applied to transform \( P(W|A) \):

\[
\hat{W} = \arg \max_w [\log(P(A|W)P(W))] \tag{2.2}
\]
where $P(A|W)$ is the likelihood of an acoustic signal given the word sequence $W$, called the *acoustic model*, and $P(W)$ is the prior probability of the sequence of words, often called the *language model*. In practice the language model probabilities are scaled by an empirically determined constant before being added to log likelihoods of an acoustic model [Young, 2008], ranging between 8 and 20. In the acoustic model, the acoustics are modelled at phone level. A word model is formed by concatenating the sequence of phones related to that word and similarly a model for the sequence of words.

The components of an ASR system are shown in Fig 2.1. A typical speech recognition system consists of an acoustic model, lexicon, language model and a decoder. A brief description of each of the modules is given in the following sections.

![Figure 2.1: Block diagram of automatic speech recognition](image)

### 2.1.1 Feature Extraction

The dynamics and the variations of acoustics across speakers and environments are modelled by acoustic features. Since speech is a dynamic signal, the features are computed over a window of 20ms duration, with an overlap of 10ms between successive windows. For decades, popular features for modelling the acoustics are Mel frequency cepstral coefficients (MFCCs) [Davis and Mermelstein, 1980] and perceptual linear prediction (PLP) coefficients [Hermansky, 1990]. Both MFCC and PLP coefficients are motivated by human perception. Usually dynamics of the speech signal across
frames are modelled by computing the first and second order temporal derivatives of static MFCC or PLP features.

2.1.2 Acoustic Model

The acoustics are modelled at phone level. The acoustic model for a word is formed by concatenating individual phone models corresponding to that word [Young, 2008], similarly for sequence of words. The parameters of phone models are estimated on training data using corresponding transcriptions. In an ASR system the phones are modelled using a hidden Markov model (HMM) [Rabiner, 1989]. An HMM consists of two stochastic processes: one stochastic process is hidden and another one is observable. The hidden stochastic process models temporal variability, whereas the observable stochastic process models the spectral variability within a state.

The parameters of an HMM are learned using the Baum-Welch algorithm [Rabiner, 1989]. Most ASR systems model context-dependent phones rather than monophones, since the acoustic realisation of a phone depends on the future and past phone realisations. Most recently, deep neural networks (DNNs) in hybrid and tandem configuration [Dahl et al., 2012; Hermansky et al., 2000; Bourlard and Morgan, 1993] and recurrent neural networks (RNNs) and variants of RNNs [Sak et al., 2014] have been used to model the acoustics.

2.1.3 Lexicon

A lexicon provides pronunciations for the words. Based on the pronunciations the phone HMMs are concatenated to form an HMM for words and sequence of words [Young, 2008]. Generally, the lexicon is prepared by human experts, i.e. linguists.

2.1.4 Language Model

The language model provides the prior probability of a sequence of words. In a typical ASR system n–grams are used in the first pass of recognition. n–grams parameters are estimated using counts of N-tuples from the training data [Young, 2008]. In the second pass of recognition either lattices or N–best lists are rescored with higher order n–grams or neural network based language models [Bengio et al., 2003].
2.1.5 Decoder

Given the acoustic model, language model and lexicon, the decoder tries to find the most likely hypothesis by searching through all possible hypotheses. Generally we use beam width to prune out unlikely hypotheses [Young, 2008].

2.2 Language Modelling

The language model estimates the probability distribution \( p(s) \) over a set of strings \( s \). A string is a sequence of words \( w_1, w_2, ..., w_N \) and the joint probability of this sequence of words \( p(w_1, w_2, ..., w_N) \) can be written as the multiplication of the conditional probabilities of each word in the sequence, as shown in equation 2.3.

\[
p(s) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)...p(w_N|w_1, w_2, ..., w_{N-1}) = \prod_{i=1}^{N} p(w_i|w_1, ..., w_{i-1})
\] (2.3)

where \( p(s) \) is the product of the probability of individual words conditioned on all the words in the context of the current word. The length of the context (number of words) increases as we approach the end of sentence. The main difficulty here is that we may not have a sufficient number of examples in the training data for a reliable estimation of probability. This is called data sparsity or the curse of dimensionality. This can be solved by conditioning the words only on previous \( n-1 \) words. The new model is called an \( n \)-gram model. After conditioning the word probability on the previous \( n-1 \) words the Equation 2.3 is approximated by Equation 2.4. The value of \( n \) of an \( n \)-gram model is referred to as its order. This terminology comes from Markov models and the \( n \)-gram model can be interpreted as a Markov model of order \( n-1 \). In the rest of this chapter we discuss how probabilities are estimated for \( n \)-grams, various smoothing algorithms for \( n \)-grams and variants of \( n \)-grams.

\[
p(s) = \prod_{i=1}^{N} p(w_i|w_1...w_{i-1}) \approx \prod_{i=1}^{N} p(w_i|w_{i-n+1}...w_{i-1})
\] (2.4)

Language models mainly differ in how the context is modelled and how the probability is estimated. In \( n \)-grams the probability is estimated in a high-dimensional discrete space based on the frequency of occurrence of words in the training data.
In neural network based approaches, the words in the context are projected into a low dimensional continuous space to predict the current word [Bengio et al., 2003]. The hidden vector representation represents the words in the context. There are several advantages to learning the probability distribution in a continuous space, which will be discussed in Section 2.5. There are two types of neural network based language models, feed-forward neural network language model (NNLM) and recurrent neural network language model (RNNLM).

In NNLMs, like $n$–grams, the context is fixed and the words are projected into a continuous space, whereas in RNNLMs the context is not fixed and the network can learn long-distance dependencies using recurrent connections (or hidden to hidden connections). RNNLMs are the current state-of-the-art for language modelling [Mikolov, 2012; Sundermeyer et al., 2012; Tran et al., 2016]. In the following sections we discuss NNLMs, RNNLMs and variants of RNNLMs.

### 2.2.1 Evaluation Metric

The metrics used to evaluate the performance of a LM are the entropy and perplexity (PPL). Entropy refers to the number of bits required to encode each word in the the test data, $T$ [Chen and Goodman, 1999]. The entropy of test data $T$ consisting of sentences $(s_1, s_2, \ldots, s_T)$ is given in Equation 2.5.

$$H(T) = -\frac{1}{N_T} \log_2 p(T)$$  \hspace{1cm} (2.5)

$$p(T) = \prod_{i=1}^{T} p(s_i)$$  \hspace{1cm} (2.6)

where $p(T)$ is the probability of the test data and $N_T$ the total number of words in the test data. The perplexity of test data $T$ is defined as the exponential of the average log probability, or entropy, of the test data, as given in Equation 2.7. The lower the entropy and PPL, the better the model is. For reporting the experimental results the PPL is preferred over the entropy [Bro; Mikolov, 2012].

$$PPL(T) = 2^{H(T)}$$  \hspace{1cm} (2.7)

$$= 2^{-\frac{1}{N_T} \log_2 p(T)}$$  \hspace{1cm} (2.8)
Chapter 2. Statistical Language Models

2.3 N–grams

In Equation 2.4, if $n=2$ the model is called a **bigram** model and if $n=3$ it is called a **trigram** model. The parameters of a **trigram** may be estimated as follows:

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})}$$  \hspace{1cm} (2.9)

where, $\frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})}$ is the maximum likelihood (ML) estimate of $p(w_i|w_{i-2}w_{i-1})$. The ML estimate always assigns highest probability to the training data of all possible **trigram** models. In general, ML estimate for an $n$–gram model is:

$$p_{ML}(w_i|w_{i-1}^{i-n+1}) = \frac{c(w_{i-n+1})}{c(w_{i-n+1}^{i-n+1})}$$  \hspace{1cm} (2.10)

where, $w_{i-n+1}$ and $w_{i-n+1}^{i-n+1}$ are the $n$–gram and the context of an $n$–gram, respectively. Here, $[c(.)]$ are the counts of occurrences of the $n$–gram and context in the training data.

The probabilities are estimated from the frequency of occurrences of an $n$–gram and its context. But there is a possibility that counts of a few $n$–grams are zero in the training data and thus their probability becomes zero. However, in speech recognition, the most likely sequence of words $\hat{W}$ that have generated the speech are selected by optimizing the Equation 2.2. If $p(w_i|w_{i-n+1}^{i-1})$ is zero in a sequence of words, $p(W)$, then $p(W|A)$ would be zero and the corresponding hypothesis will never appear in the output of a speech decoder [Chen and Goodman, 1999]. We can avoid assigning zero probabilities by using smoothing techniques. Smoothing makes the probability distributions more uniform by taking the zero probabilities upward and high probabilities downward. Smoothing reassigns probability to the zero occurring $n$–grams from the $n$–grams that occur highly in the training data.

2.3.1 Smoothing

As mentioned above, smoothing is used to assign some probability to unseen $n$–grams during testing. A number of approaches have been proposed to smooth $n$–gram probability distributions [Chen and Goodman, 1999]. The proposed approaches can be classified into **discounting**, **back-off**, **interpolation**, and also a combination of these smoothing approaches.
2.3.1.1 Discounting

In discounting, the probability distributions are adjusted such that, the zero probabilities are made greater than zero and probabilities greater than zero are discounted by a discount factor. The intuition behind discounting is that the probability for unseen events during testing is assigned with discounted probability mass from observed events. Generally, discounting is used in combination with back-off or interpolation. The proposed discounting algorithms include, Good-Turing discounting [Good, 1953], Absolute discounting [Ney et al., 1994], Witten-Bell discounting [Witten and Bell, 1991] and Kneser-Ney discounting [Kneser and Ney, 1995].

Many smoothing techniques are based on Good-Turing estimation [Chen and Goodman, 1999]. In Good-Turing smoothing, we pretend that an \( n \)-gram occurring \( r \) times in training data occurs \( r^* \) times instead,

\[
r^* = (r + 1) \frac{n_{r+1}}{n_r}
\]

where, \( n_r \) is the number of \( n \)-grams that occur \( r \) times in the training data. The count is converted into a probability by normalizing \( r^* \):

\[
p_{GT}(w_i | w_{i-n+1}) = \frac{r^*}{N}
\]

where, \( N = \sum_{r=0}^{\infty} n_r r \), so \( N \) is equal to the number of counts in the original distribution. For example, if the count of an \( n \)-gram is zero then its new count would be \( n_1/n_0 \).

Good-Turing smoothing is not used alone since it does not include the combination of higher order models with the lower models for better performance. But this smoothing approach is used in several other smoothing techniques.

In Absolute discounting, we subtract a fixed discount factor \( d \) from each non-zero count:

\[
p_{abs}(w_i | w_{i-n+1}) = \begin{cases} 
\frac{c(w_{i-n+1}) - d}{c(w_{i-n+1})} , & \text{if } c(w_{i-n+1}) > 0 \\
\alpha(w_{i-n+1}) p_{abs}(w_i | w_{i-n+2}) & \text{otherwise}
\end{cases}
\]

where, \( d \) is a discount factor that ranges between 0 to 1. The \( \alpha(w_{i-n+1}) \) normalization factor makes sure that the probability distribution for a given context \( (w_{i-n+1}) \) sums to 1. In equation 2.13, absolute discounting is in the form of back-off, but can also be used in interpolated LMs. The intuition behind absolute discounting is twofold: first, small discount values don’t significantly alter the estimates of \( n \)-grams with high counts. Second, absolute discounting modifies the estimates of \( n \)-grams with small counts, which is essential in smoothing [Chen and Goodman, 1999].
Chapter 2. Statistical Language Models

There are other smoothing techniques in which pseudo-counts are added to the existing counts. Additive smoothing or add-one smoothing is one of the simplest types of smoothing in practice. In additive smoothing we pretend that each $n$-gram occurs once more than it actually occurs in the training data. However, additive smoothing gives too much weight to non-occurring $n$-grams [Chen and Goodman, 1999].

2.3.1.2 Interpolation

Interpolation is also called Jelinek-Mercer smoothing [Jelinek and Mercer, 1980]. In this smoothing approach higher order $n$-gram models are interpolated with lower order $n$-gram models. It is useful to interpolate, because if there is insufficient data to estimate the high order model, the lower order model can provide useful information [Chen and Goodman, 1999]. The interpolated $n$-gram model is defined as:

$$ p_{\text{interp}}(w_i | w_{i-n+1}^{i-1}) = \lambda_{w_i | w_{i-n+1}^{i-1}} p_{\text{ML}}(w_i | w_{i-n+1}^{i-1}) + (1 - \lambda_{w_i | w_{i-n+1}^{i-1}}) p_{\text{interp}}(w_i | w_{i-n+2}) $$

(2.14)

The $n^{th}$-order interpolated smoothed $n$-gram model is defined as a linear interpolation between the $n^{th}$-order maximum likelihood model and the $(n-1)^{th}$-order smoothed model. Given a maximum likelihood estimate for an $n$-gram, it is possible to efficiently estimate the interpolation coefficient, $\lambda_{w_i | w_{i-n+1}^{i-1}}$, using the Baum-Welch algorithm [Rabiner, 1989]. The data used to estimate $\lambda_{w_i | w_{i-n+1}^{i-1}}$ should not be part of the training data, used to estimate $p_{\text{ML}}$. Usually the interpolation coefficients are estimated on a held-out data set, sampled from the training data.

2.3.1.3 Back-off

Back-off extends the concept of Good-Turing estimation by combining higher order models with lower order models [Katz, 1987]. It was first proposed by Katz, hence this approach is also called Katz back-off smoothing or Katz smoothing. In Katz smoothing, the probability of non-zero count $n$-grams is calculated as a discounted probability, using Good-Turing estimation [Chen and Goodman, 1999], where if the count of an $n$-gram is equal to zero it is backed-off to the corresponding $(n-1)$-gram:

$$ p_{\text{katz}}(w_i | w_{i-n+1}^{i-1}) = \begin{cases} p^*(w_i | w_{i-n+1}^{i-1}) & \text{if } c(w_i | w_{i-n+1}^{i-1}) > 0 \\ \alpha(w_{i-n+1}^{i-1}) p_{\text{katz}}(w_i | w_{i-n+2}) & \text{otherwise} \end{cases} $$

(2.15)
where, \( p^*(w_i|w_{i-n+1}^{j-1}) \) is a discounted probability estimated using the Good-Turing approach [Chen and Goodman 1999], and

\[
p^*(w_i|w_{i-n+1}^{j-1}) = \frac{c^*(w_i|w_{i-n+1}^{j-1})}{c(w_{i-n+1}^{j-1})}
\]  

(2.16)

where, \( c^*(w_i|w_{i-n+1}^{j-1}) \) is a discounted count estimated using the Good-Turing equation.

In Equation 2.15, \( \alpha(w_{i-n+1}^{j-1}) \) is the back-off coefficient for a given context \( w_{i-n+1}^{j-1} \). It is used to make sure that the probabilities sum to 1, i.e. \( \sum_w p_{katz}(w|w_{i-n+1}^{j-1}) \). For a given context, \( \alpha(w_{i-n+1}^{j-1}) \) is given by

\[
\alpha(w_{i-n+1}^{j-1}) = \frac{1 - \sum_{w:c(w_{i-n+1}^{j-1})>0} p_{katz}(w_{i-n+1}^{j-1})}{1 - \sum_{w:c(w_{i-n+1}^{j-2})>0} p_{katz}(w_{i-n+2}^{j-1})}
\]  

(2.17)

where the numerator is the discounted probability, which will be assigned to zero count \( n \)-grams and the denominator is the total probability of \( (n-1) \)-grams that have zero counts.

2.3.1.4 Kneser-Ney Smoothing

Kneser-Ney smoothing [Kneser and Ney 1995] is an extension of absolute discounting, which uses modified counts for \( m \)-gram probabilities, where \( m < n \).

\[
p_{KN}(w_i|w_{i-n+1}^{j-1}) = \frac{\max(c(w_{i-n+1}^{j-1}) - d, 0)}{c(w_{i-n+1}^{j-1})} + \frac{d}{c(w_{i-n+1}^{j-1})} N_1(w_{i-n+1}^{j-1} •) p_{KN}(w_i|w_{i-n+1}^{j-2})
\]  

(2.18)

where \( d \) is a discount factor which depends on the length of the context. \( N_1(w_{i-n+1}^{j-1} •) = |\{w' : c(w_{i-n+1}^{j-1}w') > 0\}| \), the number of distinct words \( w' \) occurring after the context, \( w_{i-n+1}^{j-1} \). Kneser-Ney smoothing was motivated by marginal constraints, the lower order probability distribution is selected such that the marginals of the higher order smoothed distribution should match the marginals of the training data.

\[
\sum_h p_{KN}(hw_i) = \frac{c(w_{i-n+2}^{j-1})}{c(w_{i-n+2}^{j-1})}
\]  

(2.19)

where \( h = w_{i-n+1}^{j-1} \). As an extension to this smoothing technique, Kneser-Ney proposed modified Kneser-Ney smoothing. In the modified Kneser-Ney smoothing technique, three different discount parameters \( d_1, d_2 \) and \( d_3 \) are applied to \( n \)-grams with count
one, two and three or more counts, respectively. However in Kneser-Ney smoothing a single discount parameter $d$ is used for all $n$–grams with non-zero counts. After modification Equation 2.18 becomes:

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \max(c(w_{i-n+1}^i) - d(c(w_{i-n+1}^i)), 0) + \gamma(w_{i-n+1}^{i-1})p_{KN}(w_i|w_{i-n+2}^{i-1})$$

(2.20)

where

$$d(c) = \begin{cases} 
0 & \text{if } c = 0 \\
1 & \text{if } c = 1 \\
2 & \text{if } c = 2 \\
3 & \text{otherwise}
\end{cases}$$

(2.21)

$$\gamma(w_{i-n+1}^{i-1}) = \frac{d_1 N_1(w_{i-n+1}^{i-1} \bullet) + d_2 N_2(w_{i-n+1}^{i-1} \bullet) + d_3 N_3 + (w_{i-n+1}^{i-1} \bullet)}{c(w_{i-n+1}^i)}$$

(2.22)

where $N_2(w_{i-n+1}^{i-1})$ and $N_3 + (w_{i-n+1}^{i-1} \bullet)$ are defined similar to $N_1(w_{i-n+1}^{i-1} \bullet)$.

Modified Knesser-Ney smoothing is motivated by the experimental observation [Chen and Goodman, 1999] that the average discount for $n$–grams with one or two counts is different from the average discount for $n$–grams with higher counts. The new discount parameters are estimated using count of count statistics:

$$d_1 = 1 - 2Y \frac{n_2}{n_1}$$

(2.23)

$$d_2 = 2 - 3Y \frac{n_3}{n_2}$$

(2.24)

$$d_3+ = 3 - 4Y \frac{n_4}{n_3}$$

(2.25)

$$Y = \frac{n_1}{n_1 + 2n_2}$$

(2.26)

where $n_r$ is the number of $n$–grams that occur $r$ times in the training data. Out of all the smoothing techniques described, modified Knesser-Ney smoothing is the state-of-the-art smoothing technique for $n$–gram language models. Based on experimental results and empirical comparisons of various smoothing techniques, a few conclusions are drawn by [Chen and Goodman, 1999] as follows:
• Out of all smoothing algorithms, interpolated and modified Kneser-Ney smoothing perform better than the others

• Interpolated models are superior to back-off models

• Adding free parameters to a smoothing algorithm and optimising them on held-out data can improve the performance

• Interpolation performs better on small training sets, whereas Katz back-off smoothing performs better on larger training sets

2.4 Beyond \textit{N}–grams

For \textit{n}–grams, probabilities are estimated from the frequency of occurrences of words in the training data. The significant advantages of \textit{n}–grams are speed of computation and generality. These models can be applied to any domain or language if we have some training data. Despite their huge success and significant advantages they have few drawbacks:

• Data Sparsity: The number of parameters of \textit{n}–grams increases exponentially with respect to the length of the context. As the number of parameters increases we need huge amounts of data for reliable estimation of these parameters. This is also called the \textit{curse of dimensionality}.

• Since the probability is estimated based on the frequency of occurrence of words, \textit{n}–grams do not consider the similarity between the words (no parameter sharing)

• Unseen sequences of words are generalised using smoothing techniques, as described in Section 2.3.1

• Due to the data sparsity issue it is also difficult to learn long-distance dependencies.

Advanced \textit{n}–gram based language models have been proposed in the literature to address some of these drawbacks. In the following sections, we describe a few such advances: Class based LM, Cache LM and Maximum entropy based LM.
2.4.1 Class Language Models

As mentioned in Section 2.4, $n$–grams suffer from data sparsity. Class based language models can address this problem [Brown et al., 1990]. The intuition is, if we assign each word to a class we can make better predictions about the histories that we haven’t seen during training. $n$–grams are trained on these classes to predict the probability of the current word. The probability of a word in class based LM is:

$$p(w_i|w_{i-n+1}^{i-1}) = p(c_i|c_{i-n+1}^{i-1})p(w_i|c_i)$$  \hspace{1cm} (2.27)

where $c_i$ is class assigned to a word $w_i$. The parameters $p(c_i|c_{i-n+1}^{i-1})$ and $p(w_i|c_i)$ can be learned from the training data using ML estimate. In class LMs, each word is assigned with a class. A class usually represents a group of words, typically syntactically or semantically related words. In [Goodman, 2001a; Mikolov et al., 2011a], words classes are formed based on frequency of occurrence of words in the training data. Class based LMs differ in the way in which the classes are formed. In soft classes, one word is assigned to multiple classes, whereas in hard classes each word is assigned to a single class. Based on experimental results the PPL improvements with class based LMs are moderate. But these models have noticeable effect on the speech recognition WER, when the training data set size is small. The disadvantage with the class based models are:

- They may require expert knowledge for the formation of the classes
- Improvements typically vanish with respect to the amount of training data [Goodman, 2001b]

Since the first work on class based LMs [Brown et al., 1990], a number of extensions have been proposed to improve them [Bellegarda et al., 1996; Gao et al., 2002; Emami and Jelinek, 2005]. In each of these works the main focus is to get better word clusters for class based LMs.

2.4.2 Cache Language Models

One of the limitation of $n$–gram models is learning long distance dependencies. Cache language models can address this problem, by incorporating the most recent words into the LM. The intuition is if a speaker uses a word, it is highly likely that he/she will use
the same word in the near future \cite{Kuhn1990, Jelinek1991, Clarkson1997}. Cache LM is also an \textit{n}–gram model but estimated dynamically from the words in the most recent context. In a typical cache LM, the smoothed \textit{n}–gram LM is interpolated with a cache \textit{n}–gram LM estimated from words in the most recent context.

\begin{equation}
\begin{aligned}
p_{\text{ngm-cache}}(w_i|w_{i-n+1}^{i-1}) &= \lambda p(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda) p_{\text{cache}}(w_i|w_{i-n+1}^{i-1})
\end{aligned}
\end{equation}

where, \( p_{\text{cache}}(w_i|w_{i-n+1}^{i-1}) \) is a cache LM estimated from words in the most recent context. It is very easy to estimate these models, for example using SRILM \cite{Stolcke2002}. Based on experimental results, it was concluded that the PPL improvements are large and significant \cite{Goodman2001b}. Since the cache models are estimated on the most recent words, speech recognition errors occurring before the current word might affect the recognition performance. So getting word error rate improvements is quite a lot harder than the perplexity improvements.

**2.4.3 Maximum Entropy Models**

Maximum entropy models have received a lot of attention since they were first used for language modelling \cite{Rosenfeld2005}. Maximum entropy (ME) models are exponential models and defined as:

\begin{equation}
\begin{aligned}
p(w_i|w_{i-n+1}^{i-1}) &= \frac{e^{\sum \lambda_k f_k(w_i,w_{i-n+1}^{i-1})}}{Z(w_{i-n+1}^{i-1})}
\end{aligned}
\end{equation}

where \( Z(w_{i-n+1}^{i-1}) \) is a normalisation term used to normalise the probability distribution:

\begin{equation}
\begin{aligned}
Z(w_{i-n+1}^{i-1}) &= \sum_{w \in V} e^{\sum \lambda_k f_k(w,w_{i-n+1}^{i-1})}
\end{aligned}
\end{equation}

where \( V \) is the vocabulary of the speech recognition task. This model can be used as a model that combines features, \( f_k(w,w_{i-n+1}^{i-1}) \), which are computed from the training data. Learning involves finding an optimal set of parameters \( \lambda_k \) and a good set of features. The \( \lambda_k \)s are learned using the Generalized Iterative Scaling (GIS) algorithm \cite{Darroch1972}. Generally ME models are trained with \textit{n}–gram, skip-gram and syntactic and semantic features \cite{Khudanpur2000}. The power of these models comes from adding features and learning the models from many features. These models have been shown to provide significant improvements in perplexity and word error rates, when trained on triggers and word features \cite{Rosenfeld2005}. One
of the limitation is these models is that they are extremely time consuming to train and test [Goodman, 2001b]. Since [Rosenfeld, 2005], a lot of work has been done to improve these models further. One of the most notable works is \textit{model–M}, which is a class based exponential model [Chen, 2009]. [Chen, 2009] reported significant reductions in word error rates on Broadcast speech recognition task and also showed that \textit{model–M} can scale well and works across domains. Most recently, [Mikolov, 2012] showed that ME models can be trained using stochastic gradient decent (SGD) by back-propagating the errors. Since it can be trained with SGD, this model can be viewed as a neural network based language model without a hidden layer.

2.5 Neural Network Language Models

As discussed in Section 2.3, \textit{n}–gram parameters are estimated based on the frequency of occurrence of words in training data. In class based LMs, as described in Section 2.4.1, each word is assigned to a class to increase the generalisation ability of an \textit{n}–gram LM to unseen sequences of words during testing. However, performance mainly depends on class assignment. Other approaches have been proposed to increase the generalisation of a LM to unseen sequences of words. But these approaches are based on neural networks and distributed representations [Xu and Rudnicky, 2000; Bengio et al., 2001; Schwenk and Gauvain, 2002; Bengio et al., 2003]. Neural networks are currently state-of-the-art in language models. Most recently, recurrent neural network language models and their variants have shown significant and consistent improvements across the tasks [Mikolov et al., 2010, 2011a; Mikolov, 2012; Sundermeyer et al., 2012].

For \textit{n}–grams, the curse of dimensionality is the fundamental problem that makes it difficult to train and generalise a LM to an unseen sequence of words. For example, if the vocabulary is 10K and we want model the joint distribution of 10 consecutive words, then the number of free parameters would be $10^{40} - 1$: we would need huge amounts of data for reliable estimation of these parameters. In \textit{n}–grams, the probability is estimated in a discrete space whereas in models based on distributed representations (e.g., Neural networks) the probability is estimated in continuous space. In continuous space, generalisation is much easier than in discrete space. The intuition is that, in continuous space the probability estimators are smooth functions of distributed representations. Given the relative ease of generalisation in continuous space, approaches based on distributed representations have been proposed. In neural network language mod-
nels, each word is assigned with a low dimensional distributed representation. These distributed representations are learned by projecting the words into continuous space.

In addition to the curse of dimensionality and generalisation, the other drawback of \( n \)-grams is that \( n \)-grams do not consider the similarity between the words. In neural networks each word is assigned with a feature vector, so similarities (syntactic and semantic) between the words can be exploited. For example the following training data examples:

- A student is giving a talk on Monday
- A Professor gave a lecture on Friday
- A Lecturer is giving a talk on Thursday

In this example, similar words (semantic and syntactic) are assigned with similar features vectors. So, the existence of one of the above sentences in the training data will increase the probability of that sentence and also its neighbours (semantic and syntactic) in sentence space. The other advantage is, the number of parameters of the model scales linearly with respect to the order of the model, \( n \). There are two types of neural network language models: feed-forward and recurrent neural network language models. In the following sections we describe both of them.

### 2.5.1 Feed-forward Neural Network Language Models

The architecture of a feed-forward neural network language model (NNLM) is given in Figure 2.2. During training NNLM has to perform two tasks. First, it has to learn the distributed representations for each word in the vocabulary. Second, it has to estimate the probability of the word \( w_i \) given its context \( w_{i-n+1}^{i-1} \), \( p(w_i|w_{i-n+1}^{i-1}) \). The parameters of the network are learned using stochastic gradient descent, either in on-line or batch mode.

The inputs to the network are indices of previous words in the context, \( w_{i-n+1}^{i-1} \). These words are encoded using a so-called 1-of-\( N \) coding, where \( N \) is the size of vocabulary. Typical values of \( N \) range from 50K to 200K. A 1-of-\( N \) coding of word \( w_i \) is an \( N \) dimensional binary vector where the \( i^{th} \) word of the vocabulary is coded by setting the \( i^{th} \) element of vector to 1 and all other elements to zero. This type of encoding simplifies the computations in the projection layer, we just need to copy the
\[ p(w_i = 1 | w_{i-1}^{i-1}) \]
\[ p(w_i = 2 | w_{i-2}^{i-1}) \]
\[ p(w_i = 3 | w_{i-2}^{i-1}) \]
\[ p(w_i = 4 | w_{i-2}^{i-1}) \]
\[ p(w_i = k | w_{i-2}^{i-1}) \]
\[ p(w_i = N | w_{i-2}^{i-1}) \]

Figure 2.2: Feed-forward neural network language model [Bengio et al., 2003; Schwenk, 2007]

The \( i \)th row of the weight matrix connecting the input and the projection layer, an \( N \times P \) projection matrix, where \( P \) is the dimensionality of the projection or distributed representation. The projection matrix is shared across all the words in the context. A linear activation function is used in the projection layer. The distributed representations of words in the context \( (w_{i-n+1}^{i-1}) \) are concatenated to form the input representation for the hidden layer of the network. Let's denote the activations of this layer with \( x_l \) with \( l = 1, 2, \ldots, n \).

Given the concatenated representation of words, the remaining two layers estimate the probability of word \( w_i \) given the context. At the hidden layer, the representation is computed by applying a non-linear hyperbolic tangent function:

\[
 h_j = \tanh \left( \sum_{l=1}^{(n-1)P} v_{jl}x_l \right) \quad \forall j = 1, \ldots, H
\]

where \( v_{jl} \) is the weight matrix between projection and hidden layer, and \( h_j \) is the hidden activation for the \( j \)th hidden unit, after applying a tanh non-linearity. The output layer is a softmax layer [Bridle, 1990]. In the output layer NNLM computes the probability
of all words given the context, \( w_{i-n+1} \),

\[
o_k = \sum_{j=1}^{H} w_{kj} h_j \quad \forall k = 1, \ldots, N
\]  

(2.32)

\[
p_k = \frac{e^{o_k}}{\sum_{l=1}^{N} e^{o_l}}
\]  

(2.33)

where \( w_{kj} \) are the weights between the hidden and output layer. \( o_k \) are the output layer activations before softmax operation. \( p_k \) is the output of neuron \( k \), which directly corresponds to the probability \( P(w_i = k | w_{i-n+1}^{i-1}) \). In Equation 2.33, the denominator is a normalisation term which needs to be computed each time, during training and testing, whereas for \( n \)-grams the normalisation term is computed during training and the computed value used during testing.

The objective of training is to optimise the cross-entropy (CE) between the target and output probability distributions.

\[
CE = - \sum_{k=1}^{N} t_k \log p_k
\]  

(2.34)

where \( t_k \) and \( p_k \) are desired and actual output of neuron \( k \), respectively. Usually \( t_k = 1 \) for target word and zero for all the other words. During training the objective is to minimize the cross-entropy by iterating several times over the training data. Generally, cross-entropy on a held-out/validation dataset is used for early stopping training or to avoid overfitting. The parameters of the network are learned by using stochastic gradient decent (SGD) [Rumelhart et al., 1986]. In SGD the errors computed in the output layer are propagated back to update the weights of the network. The weight updates can be done either on-line or batch mode. In on-line mode the weights are updated after presenting each example to the network, whereas in batch mode the weights are updated after processing a subset of examples in training data.

The computational complexity of the model increases with respect to the size of the vocabulary. Let’s analyse the computational complexity of the entire model. The computation in the projection layer is almost zero, since computing a distributed representation is just a lookup in the projection matrix. The number of computations (multiply and add) between the projection and hidden layer are \( (P.(n-1) \times H) \), where \( P \) is the dimensionality of projection. Similarly the number of computations between the hidden layer and output layer is \( (H \times N) \). The total number of number of computations including element-wise non-linearity and softmax are:

\[
(P.(n-1) \times H) + H + (H \times N) + N
\]  

(2.35)
The computational bottleneck is the number of computations in the output layer. As the vocabulary size, \( N \), increases the number of computations increases accordingly. In LVCSR systems, typical values for \( N \) are in the range of 50K to 150K. In the literature a number of techniques have been proposed to reduce the computational complexity of the output layer. These include techniques based on noise contrastive estimation [Chen et al., 2015b; Mnih and Teh 2012], importance sampling [Bengio and Senecal, 2003] and structuring the vocabulary into tree-like structure [Morin and Bengio 2005; Le et al., 2011].

2.5.2 Recurrent Neural Network Language Models

In feed-forward neural network language models (NNLMs) the length of the context is fixed: the previous \( n - 1 \) words. It is obvious to think about models which can learn contexts of arbitrary length implicitly during training. In Recurrent neural network language models (RNNLM) the length of the context is not fixed. Ideally, the RNNLMs can learn contexts of infinite length. But due to the vanishing and exploding gradient problem [Bengio et al., 1994], the length of the context learned by RNNLM is not infinite. Using recurrent or hidden-to-hidden connections the context information can cycle through the network for a very long time. So it can effectively learn long distance dependencies and complex patterns in sequential data [Mikolov, 2012].

The architecture of RNNLM is shown in Figure 2.3. The inputs to the network are the previous word and the state of the hidden layer at \( t - 1 \), \( h_{t-1} \). The previous word is encoded using 1-of-\( N \) coding. The hidden layer uses a sigmoid activation function to control the range of parameters.

\[
h_j(t) = f\left( \sum_{l=1}^{N} u_{jl} x_l + \sum_{l=1}^{H} v_{jl} h_l(t-1) \right) \quad \forall j = 1, ..., H
\]  

(2.36)

where, \( u_{jl} \) and \( v_{jl} \) are elements of input-to-hidden and hidden-to-hidden matrices, respectively, \( f \) is a sigmoid activation function, and \( h(t-1) \) and \( h(t) \) are the activations of hidden layer at \( t-1 \) and \( t \), respectively. The output layer is a softmax layer. In the output layer, the RNNLM computes the probability of word \( w_i \) given the previous word \( w_{i-1} \) and context, \( h_{t-1} \).

\[
o_k = \sum_{j=1}^{H} w_{kj} h_j(t) \quad \forall k = 1, ..., N
\]  

(2.37)

\[
p_k = \frac{e^{o_k}}{\sum_i e^{o_i}}
\]  

(2.38)
where $p_k$ represents the probability of the word given the context, $P(w_i = k|h(t - 1), w_{i-1})$.

The computational complexity is approximately equal to:

$$H \times H + H \times N$$

(2.39)

where $N$ is the vocabulary size and $H$ is the number of hidden neurons. As discussed in Section 2.5.1, the computational bottleneck is the size of the output layer or vocabulary size. In addition to the size of the output layer the other bottleneck is the number of time steps that we propagate the error back in time, this method of training is called back propagation through time (this will be discussed in the following section). If we propagate the errors back in time at each time step this would also increase the total number of computations.

### 2.5.2.1 Back propagation through time

The RNNLMs are trained by stochastic gradient descent using either back propagation (BP) or the back propagation through time (BPTT) algorithm, which is a slight...
modification to the back propagation (BP) algorithm. In BP, the current word is predicted given the previous word and state of the hidden layer at time $t-1$. Given that the RNNLM has recurrent connections, training using BP is not optimal since the hidden state is not storing any information from the past words which will be useful in the near future. In BPTT, the errors computed are propagated back in time to update the weights at time $t$. By propagating the errors back in time the hidden state of the RNNLM learns useful long term context information to predict the current word. Since BPTT makes the RNNLM learn long distance dependencies, the accuracy of a RNNLM trained using BPTT is better than one trained using BP.

The inputs to the network are the previous word (encoded using $I$-of-$N$ coding) and the hidden layer activations at time $t-1$. The objective is to minimize the cross-entropy between the target and output probability distributions. In other words, to maximize the likelihood of the training data. The cross-entropy between the output and target probability distributions is defined as:

$$CE = - \sum_{k=1}^{N} t_k \log p_k$$  \hspace{1cm} (2.40)

where $t_k$ and $p_k$ are desired output and target outputs a neuron $k$, respectively. Usually $t_k = 1$ for target word and zero for all the other words. In BP, the errors in the output layer are computed by computing the gradient of CE with respect to $o_k$:

$$e_{ok}(t) = \frac{\partial (CE)}{\partial o_k} \quad \forall \quad k = 1,...,N$$  \hspace{1cm} (2.41)

$$e_{ok}(t) = t_k - o_k$$  \hspace{1cm} (2.42)

where $e_{o}(t)$ are errors at the output layer.

The errors in the output layer are used to update the weights between the hidden and output layer.

$$w_{jk}(t) = w_{jk}(t-1) + \alpha h_j(t)e_{ok}(t) \quad \forall j = 1,...H \quad and \quad \forall k = 1,...,N$$  \hspace{1cm} (2.43)

where $\alpha$ is the learning rate. After updating the weights, the output layer errors are propagated back to the hidden layer. Errors at the hidden layer are:

$$e_{hj}(t) = \sum_{k=1}^{N} w_{jk}e_{ok}(t) \quad \forall j = 1,...H$$  \hspace{1cm} (2.44)

$$e_{hj}(t) = e_{hj}(t)h_j(t)(1-h_j(t))$$  \hspace{1cm} (2.45)
2.5. Neural Network Language Models

where \(e_{hj}(t)\) are the errors at the hidden layer. The errors at the hidden layer are used to update the input to the hidden \((U)\) and recurrent weights \((V)\) as follows:

\[
u_{jl}(t) = u_{jl}(t - 1) + \alpha e_{hj}(t) x_{l}(t) \quad \forall j = 1,\ldots,H \text{ and } l = 1,\ldots,N \quad (2.46)
\]

\[
v_{jl}(t) = v_{jl}(t - 1) + \alpha e_{hj}(t) h(t - 1) \quad \forall j = 1,\ldots,H \text{ and } i = 1,\ldots,H \quad (2.47)
\]

As mentioned above, in the normal BP algorithm, the RNNLM is trained in the same way as a feed-forward NNLM. The only difference is that the current hidden state depends on the previous hidden state. If trained using BPTT, the hidden state can store information about the words in the distant past and learn long-distance dependencies, which are essential to improve prediction accuracy.

In BPTT, the RNNLM used to model a sequence of \(N\) words can be seen as a deep neural network with \(N\) hidden layers with shared hidden-to-hidden layer connections, as shown in Figure 2.4.

In BPTT, errors at hidden layer, \(e_{hj}(t)\) are propagated to the hidden layer at time \(t - 1\). The errors at \(h(t - 1)\) are computed as follows:

\[
e_{hj}(t - 1) = \sum_{l=1}^{H} v_{jl} e_{hl}(t) \quad \forall j = 1,\ldots,H \quad (2.48)
\]

where \(e_{hj}(t - 1)\) are errors at hidden layer of \(t - 1\). Once the errors are propagated back in time, the recurrent weights \((v_{jl})\) are updated. Similarly the errors at time \(t - 1\) are propagated back in time recursively until the start of the sequence as follows:

\[
e_{hj}(t - \tau - 1) = \sum_{l=1}^{H} v_{jl} e_{hl}(t - \tau) \quad \forall j = 1,\ldots,H \quad (2.49)
\]
where \( \tau \) runs from 0 to number of time steps that the network is unfolded back in time. However, as we propagate the errors back in time, these errors may vanish or explode. This is called the vanishing or exploding gradient problem [Bengio et al., 1994]. A simple solution is to truncate the number of time steps that we propagate the errors back in time. Empirical observations have revealed that, propagating the errors 4 to 5 time steps back in time is sufficient to store the required information in the hidden state of the RNNLM [Mikolov, 2012].

In BPTT, the weight updates of input-to-hidden and hidden-to-hidden connections will change as follows:

\[
 u_{jl}(t) = u_{jl}(t-1) + \sum_{\tau=0}^{T} x_l(t-\tau)e_{h_j}(t-\tau)\alpha \quad \forall j = 1, \ldots, H \text{ and } l = 1, \ldots, N \quad (2.50)
\]

where \( T \) is the number of time steps that the network is unfolded back in time. Similarly, the updates to the recurrent connections will change as follows:

\[
 v_{jl}(t) = v_{jl}(t-1) + \sum_{\tau=0}^{T} h_j(t-\tau-1)e_{h_j}(t-\tau)\alpha \quad \forall l = 1, \ldots, H \text{ and } j = 1, \ldots, H \quad (2.51)
\]

### 2.5.3 Long-Short Term Memory Recurrent Neural Networks

As described in Section 2.5.2.1, it is difficult to train RNNLMs using the back propagation through time (BPTT) algorithm. The main problem is due to the vanishing and exploding gradient problem [Bengio et al., 1994]. In BPTT, while propagating the errors back in time, these errors may vanish or explode. This is called the vanishing or exploding gradient problem. A simple solution is to truncate the number of time steps that we propagate the errors back in time. Empirical observations have revealed that, propagating the errors 4 to 5 time steps back in time is sufficient to store the required information in the hidden state of the RNNLM [Mikolov, 2012].
errors back in time either they vanish or explode. The explanation for the vanishing and exploding gradient problem is clearly explained in [Hochreiter and Schmidhuber, 1997]. Whenever the error is propagated back through the neurons, the errors get scaled by a factor of more or less than one. As a result of scaling, sometimes the errors may vanish or explode. Long-short term memory (LSTM) neural networks were proposed to address the vanishing and exploding gradient problem [Hochreiter and Schmidhuber, 1997].

In LSTMs, the neuron is re-designed in such a way that the scaling factor is fixed to one. Since the scaling factor is one, the learning capabilities of the re-designed unit are limited. The learning capabilities are enriched by adding several gating units. The final LSTM neuron, after adding gating units, is shown in Figure 2.5. In Figure 2.5 the input, output and forget gates are depicted as blue circles whereas the memory cell is depicted as a red circle. The content of the memory cell is controlled by the output of the forget gate. If the forget gate output is zero then the content of the memory cell is replaced with a new representation equivalent to the current context. Peep-hole connections (represented by red arrows, $c_{t-1}$) were introduced in later versions of LSTM architectures [Gers and Schmidhuber, 2000]. Peep-hole connections allow the gates to see the state of the memory cell. At time $t$, the activations of gates, memory cell and LSTM unit are computed as follows:

\[
i_t = \sigma(w_{xi}x_t + w_{hi}h_{t-1} + w_{ci}c_{t-1} + b_i) \tag{2.52}
\]

\[
f_t = \sigma(w_{xf}x_t + w_{hf}h_{t-1} + w_{cf}c_{t-1} + b_f) \tag{2.53}
\]

\[
c_t = f_tc_{t-1} + i_t * \tanh(w_{xc}x_t + w_{hc}h_{t-1} + b_c) \tag{2.54}
\]

\[
o_t = \sigma(w_{xo}x_t + w_{ho}h_{t-1} + w_{co}c_{t-1} + b_o) \tag{2.55}
\]

\[
h_t = o_t * \tanh(c_t) \tag{2.56}
\]

where, $x_t$, $h_{t-1}$ and $c_{t-1}$ are respectively the input at time $t$, the state of the hidden layer at $t-1$, and state of the memory cell at time $t-1$. $i_t$, $f_t$ and $o_t$ are the outputs of input, forget and output gates respectively. $c_t$ is content of memory cell at time $t$. $\sigma(.)$ and $\tanh(.)$ are sigmoid and tanh activation functions respectively. More details about back propagation of errors and weight update of LSTMs are given in [Graves and Schmidhuber, 2005]. From the experimental results, it has been observed that LSTMs outperform the simple RNNs in terms of WERs and PPLs [Sundermeyer et al., 2012; Jozefowicz et al., 2016].
2.5.4 Gated Recurrent Neural Networks

One of the variations of LSTM that has become very popular in recent days is the Gated Recurrent Unit (GRU) [Cho et al., 2014]. In a GRU, the input and forget gates are combined to form a new gate called the update gate. Changes have also been made in terms of merging the hidden cell states. GRUs are simpler than LSTMs and have been applied to machine translation [Cho et al., 2014], sequence modelling [Chung et al., 2014] and, most recently, language modelling [Kazuki Irie et al., 2016]. Please refer to [Chung et al., 2014], for more information regarding the computation of hidden activations, back propagation of errors and weight update in GRUs.

2.5.5 Recurrent Neural Networks with Attention

Recently, recurrent neural networks with attention have been successfully applied to various tasks like handwriting synthesis [Graves, 2013], image caption generation [Xu et al., 2015], machine translation [Bahdanau et al., 2014] and speech recognition [Chorowski et al., 2015]. In attention based models the input is iteratively processed to select the relevant part of the input to predict the current target in the sequence. This idea has also led to the development of networks based on external memory [Graves et al., 2014; Weston et al., 2014]. Most recently attention based approaches have been used for language models [Tran et al., 2016].

2.6 Summary

In the first part of this chapter we described the automatic speech recognition system and its components. In the second part, we defined the language modelling problem. In the third part, description of n-grams, evaluation metrics, various smoothing techniques and advanced n-gram models (cache, class and max-entropy) were discussed. In the final part, we described NNLMs, RNNLMs and LSTMs and their variants.
Chapter 3

Datasets and Experimental Setup

3.1 Introduction

In the first part of the chapter we describe various data sets used to train and evaluate the proposed language models (LMs) for automatic speech recognition (ASR). In the second part of the chapter a description of the experimental setup used to incorporate the proposed LMs into the ASR is given.

The rest of the chapter is organised as follows: In section 3.2.1 we describe the Penn Treebank data set. The resources available for TED and Multi-Genre Broadcast speech recognition tasks are given in Section 3.2.2 and Section 3.2.3 respectively. Finally, the description of the N-best list rescoring experimental setup is given in Section 3.3.

3.2 Datasets

In this thesis, datasets in English are used to evaluate the language models.

3.2.1 Penn Treebank Corpus

The Penn Treebank [1] (PTB) corpus is widely used to evaluate the performance of language models prior to word error rate (WER) experiments. This corpus was created by taking a portion of utterances from the WSJ corpus [Paul and Baker, 1992]. The PTB corpus has been used by many researchers with the same preprocessing and vocabulary (limited to 10K most frequent words). Since it has been widely used, it has the advan-

[1] https://www.cis.upenn.edu/~treebank/
tage of allowing us to compare the performances of proposed LMs with other LM techniques and their combinations. In Table 3.1 we give the best perplexities reported on this dataset so far in the literature.

<table>
<thead>
<tr>
<th>Model</th>
<th>test PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNNLM ([Mikolov, 2012])</td>
<td>124.7</td>
</tr>
<tr>
<td>RNNLM-LDA ([Mikolov and Zweig, 2012])</td>
<td>113.7</td>
</tr>
<tr>
<td>genCNN ([Wang et al., 2015])</td>
<td>116.4</td>
</tr>
<tr>
<td>FOEE-FNNLM ([Zhang et al., 2015])</td>
<td>108</td>
</tr>
<tr>
<td>Sum-Product Net ([Cheng et al., 2014])</td>
<td>100</td>
</tr>
<tr>
<td>LSTM-char([Kim et al., 2015])</td>
<td>78.9</td>
</tr>
<tr>
<td>large-regularized-LSTMs ([Zaremba et al., 2014])</td>
<td>68.7</td>
</tr>
</tbody>
</table>

Table 3.1: Best perplexities (PPL) reported so far on PTB test data in the literature

The PTB dataset is divided as follows: Sections 0–20 were used as training data, sections 21–22 were used as validation data, and sections 23–24 used as test data. The vocabulary is the 10K most frequent words from the training data. All the out of vocabulary words are mapped to a special token <unk>. The number of tokens in the training, validation and test sets are given in Table 3.2.

<table>
<thead>
<tr>
<th>Data</th>
<th>#tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>930K</td>
</tr>
<tr>
<td>valid</td>
<td>74K</td>
</tr>
<tr>
<td>test</td>
<td>82K</td>
</tr>
</tbody>
</table>

Table 3.2: Number of tokens in train, valid and test set of PTB corpus

### 3.2.2 TED lecture transcription task

TED (Technology, Entertainment, Design) organises an international lecture series in a wide range of disciplines. The lectures are transcribed (and translated) by volunteers, providing about 200 hours of transcribed data, which have been used for ASR and machine translation evaluation as part of the IWSLT evaluation campaign.

[2](http://www.ted.com) [3](http://www.iwslt2013.org)
3.2. Datasets

3.2.2.1 Acoustic model data

We used 813 TED lectures recorded prior to end of 2010. The lectures were segmented and aligned to transcripts using a lightly-supervised technique [Stan et al., 2012]. This alignment produced 143 hours of transcribed speech for acoustic model training (in-domain data for transcription task).

3.2.2.2 Language model data

<table>
<thead>
<tr>
<th>Corpus</th>
<th>#tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europarl</td>
<td>53.1M</td>
</tr>
<tr>
<td>News Commentary</td>
<td>4.4M</td>
</tr>
<tr>
<td>News Crawl</td>
<td>693.5M</td>
</tr>
<tr>
<td>GigaWord</td>
<td>2915.6M</td>
</tr>
<tr>
<td>OOD Total</td>
<td>3666.6M</td>
</tr>
</tbody>
</table>

Table 3.3: Number of tokens in various OOD data sources

To train the language models, the IWSLT evaluation provided 2.4M tokens of in-domain TED lecture data and various larger out-of-domain (OOD) data sources. The OOD data sources were Europarl (v7), News Commentary (v7), News Crawl (2007 to 2011) and the fifth version of Gigaword. The number of tokens in in-domain and various OOD sources are given in Table 3.3.

The text from data sources like News Crawl and Gigaword include phenomena such as numerical expressions, abbreviations, money amounts and tabulated information. Text normalisation is required to convert these phenomena into spoken word sequences. Text normalisation is done using in-house tool\textsuperscript{4}. The normalisation steps applied to the data can be summarised as follows:

- Eliminate duplicate lines (common in newswire sources, where multiple copies or variants of the same story occur)

- Remove documents that are not of type *story*, strip out markup and split text into sentences (required for Gigaword only).

- Convert encodings for fractions and symbols and unicode characters

\textsuperscript{4}Thanks to Fergus McInnes, for writing perl scripts to normalise the text [Bell et al., 2013a]
• Normalise punctuation and abbreviations

• numerical expressions to words

• Convert upper-case to lower-case

• Correct spelling errors and British-to-American English spellings

Given the mismatch in content and style between the in-domain and OOD data, a data selection process \cite{Yamamoto:2012, Moore:2010} is applied to the OOD data to obtain a subset of data for LM training. This subset of OOD data consists of sentences relevant to the task domain. In data selection process sentences were scored based on cross-entropy difference between the models trained on in-domain and random subset of OOD data. We choose OOD sentences for which the cross-entropy difference is below a threshold ($\tau$). This threshold was optimized on held-out dataset. The cross-entropy difference (CED) score for OOD sentences is computed as follows:

$$D_S = \{s | H_I(s) - H_O(s) < \tau \}$$

(3.1)

where, $D_S$ is cross-entropy score. $H_I(s)$ is a cross-entropy of a sentence with a LM trained on in-domain data, $H_O(s)$ is a cross-entropy of a sentence with a LM trained on a random subset of the OOD data of similar size to the TED in-domain data, and $\tau$ is a threshold to control the size of $D_S$.

Using the data selection process, we selected a total of 312M tokens from the OOD data to train the LMs. The LMs finally used in the decoding and lattice rescoring were trained on TED in-domain data and the selected 312M OOD subset. The number of tokens selected from each OOD data source to created subset of OOD are given in Table 3.4.

The vocabulary of this task was defined to include the all the words occurring in in-domain data (except the words which occurred only once) and all the words exceeding specified occurrence count thresholds in the OOD corpora. The vocabulary size after including words from in-domain and OOD data was 62,522. The $n$-gram LMs were trained using SRILM toolkit \cite{Stolcke:2002}.

For this task, we used the dev2010 and tst2010 sets for development and the tst2011 set for evaluation. The out-of-vocabulary (OOV) rates of dev 2010, tst2010 and tst2011 are given in Table 3.5.
### 3.2. Datasets

<table>
<thead>
<tr>
<th>Corpus</th>
<th>original</th>
<th>selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europarl</td>
<td>53.1M</td>
<td>6.3M</td>
</tr>
<tr>
<td>News Commentary</td>
<td>4.4M</td>
<td>0.7M</td>
</tr>
<tr>
<td>News Crawl</td>
<td>693.5M</td>
<td>72.9M</td>
</tr>
<tr>
<td>GigaWord</td>
<td>2915.6M</td>
<td>232.9</td>
</tr>
<tr>
<td>OOD Total</td>
<td>3666.6M</td>
<td>312.8M</td>
</tr>
</tbody>
</table>

Table 3.4: Number of tokens in various OOD data sources before and after the data selection process

<table>
<thead>
<tr>
<th>Data set</th>
<th>OOV rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev2010</td>
<td>1.23</td>
</tr>
<tr>
<td>tst2010</td>
<td>0.69</td>
</tr>
<tr>
<td>tst2011</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Table 3.5: OOV rates of development and test data sets of TED task

#### 3.2.3 Multi-Genre Broadcast (MGB) Speech Recognition Task

The Multi-Genre Broadcast (MGB) challenge [Bell et al., 2015](http://www.mgb-challenge.org/) was part of the official challenges at ASRU 2015. The data for the MGB challenge was provided by the British Broadcast Corporation (BBC) and consists of BBC TV programmes. The provided dataset is very broad and consists of programmes from multiple genres. The BBC provided a total of 1600 hours of acoustic data and several hundred million words of subtitle text for language models.

The MGB challenge consists of a total of four tasks. Out of these four tasks, two are transcription tasks. One is a standard ASR task and the second is a longitudinal transcription task. But in this thesis we evaluate the proposed language models in the context of a standard ASR task.

#### 3.2.3.1 Metadata

The subtitles provided with audio consist of transcripts and other metadata. The metadata includes speaker changes, time stamps, the title, date, TV channel and genre of each show. The quality of the metadata varies across genres, in terms of preci-
Chapter 3. Datasets and Experimental Setup

3.2.3.2 Acoustic Data

The MGB challenge provided the participants with 7 weeks of BBC television programmes (from BBC1, BBC2, BBC3, BBC4). These programmes are BBC output over the period 1 April 2008 to 19 May 2008. In addition to the acoustic data the MGB challenge also provided subtitles for the audio.

Prior to training the acoustic models on audio data, a two step refinement process was applied. In the first step subtitles are aligned to audio to extract the exact time boundaries and in the second step different measures were computed to select the training data. The alignment procedure was based on the lightly supervised approach described in [Braunschweiler et al.; Lanchantin et al., 2013]. The data segments for acoustic model training were selected based on two error rate measures, phone matched error rate (PMER)/ matched error rate (MER) and the average word duration (AWD). Further details about data selection can be found in [Bell et al., 2015].

3.2.3.3 Language Model Data

The MGB Challenge provided a 640M token corpus of language model training data containing BBC subtitle data recorded during 1979–2013, all obtained from pre-recorded (rather than live) subtitling. The acoustic training data for the MGB Challenge includes about 10M transcribed words (from both live and prerecorded subtitles): this data was not used to train the baseline language models, but was only used for supervised adaptation experiments (Table 5.3). Before training the LMs the text data was normalised, with numbers converted to text-form and abbreviations converted to sequences of letters.

The MGB Challenge development set comprised 47 shows (28 hours of audio), called dev.full. In this work the parameters of n–gram and RNNLM are tuned on dev.full. A total of four eval test sets are available for different tasks. In this thesis we used the eval test set for the transcription task, eval.task1, which consists of 16 shows (11 hours of audio).
3.3 Experimental Setup

In LVCSR systems, LMs are incorporated into ASR during decoding or to rescore the lattices or to rescore N–best lists. In general, pruned n–grams are used in the decoding step. Later in the second pass of the recognition, lattices are rescored using strong LMs (usually with a 4–gram or 5–gram). However neural network based LMs are incorporated into the ASR during second pass recognition (lattice or N–best list rescoring). Since the context is fixed in NNLMs, like n–grams these models also can be straightforwardly incorporated into ASR during first pass decoding.

As described in Section 2.5.2, RNNLMs learn long distance dependencies using recurrent connections and the hidden vector represents the words it has seen so far. So the context used to predict the current word is dynamic and variable. Due to this we cannot use the RNNLMs in first pass decoding because introducing it in first pass decoding increases the number of computations exponentially with respect to the length of the sequence.

However, recently RNNLMs have been used in first pass decoding by approximating the context used to predict the current word [Hori et al., 2014]. In [Liu et al., 2016b], lattices are rescored using RNNLMs by approximating the contexts. In this thesis we incorporated RNNLMs into ASR by rescoring N–best lists. The N–best lists are generated by selecting N–best paths from a lattice. The N-best hypothesis (linear form of a lattice) for a given utterance is generated from a lattice. Each path in the lattice represents a hypothesis of an utterance.

The experimental setup to rescore the N–best lists is shown in Figure 3.1. As shown in Figure 3.1 in first pass decoding the decoder uses an acoustic model, lexicon and pruned n–gram LM to generate the lattice of each utterance in the test data. In the second pass, N-best lists generated from the lattice are rescored using strong/unpruned n–grams or neural network based LMs. As discussed above, due to computational complexity, it is recommended to incorporate the neural network based models like RNNLMs into ASR by rescoring the N–best lists. Usually 100–best hypotheses of an utterance are generated from the lattice for rescoring. Since neural network based LMs (NNLMs/RNNLMs) are complementary to standard n–grams, the scores of neural network based LMs are interpolated with the scores of n–grams, before selecting the final 1-best.
Figure 3.1: $N$–best list rescoring setup

### 3.4 Summary

In this chapter we described the TED lecture transcription task and the MGB challenge. Description also included the data and available resources to train acoustic and language models. We also described the experimental setup used to incorporate the RNNLMs into ASR.
Chapter 4

Feed-forward Pre-training for Recurrent Neural Network Language Models

In this chapter, we propose a pre-training method for the recurrent neural network language model (RNNLM), by sharing the output weights of a feed-forward neural network language model (NNLM) with the RNNLM. This is accomplished by first fine-tuning the weights of the NNLM, which are then used to initialise the output weights of an RNNLM with the same number of hidden units. We have carried out text-based experiments on the Penn Treebank Wall Street Journal data (described in Chapter 3), and ASR experiments on the TED talks data (described in Chapter 3). Across the experiments, we observe small but significant improvements in perplexity and ASR word error rate.

4.1 Introduction

The main goal of this chapter is to investigate whether transfer of information from a trained feed-forward neural network can be used to initialise the parameters of an RNNLM. In [Sutskever et al., 2013], the importance of initialisation when training RNNs is highlighted, setting the scale of the input weights such that the learning dynamics of the RNN neither quickly “forget” the hidden state, nor cause the error gradients to explode. Given the importance of parameter initialisation, in this chapter, we have conducted experiments to answer four questions. The first question is will pre-training the parameters of RNNLM help to improve the prediction accuracy? Second,
will the pre-training reduce the number of training iterations required to converge to a local minimum? Third, how many pre-training iterations are necessary to improve the prediction accuracy? Finally, if prediction accuracy improves, will it improve the speech recognition word error rates (WERs)?

We verify the first hypothesis based on perplexity (PPL) experiments on the Penn Treebank dataset (described in Section 3.2.1). The lower the PPL the better the model with improved prediction accuracy. We used the same Penn Treebank dataset to verify the second and third hypotheses also. Finally, the fourth hypothesis is verified by incorporating the pre-trained RNNLM into ASR by rescoring the $N$-best lists (experimental setup is described in Chapter 3). We conducted ASR-based experiments using the TED talks task (described in Chapter 3) that we previously investigated as part of the International Workshop on Spoken Language Translation (IWSLT) evaluation campaigns [Hasler et al., 2012; Bell et al., 2013a].

In rest of the chapter we describe the algorithm for the proposed pre-training for RNNLMs, related work and the experimental results.

### 4.2 Related Work

Prior to 2006 training neural networks with multiple hidden layers was considered to be difficult. An explanation for this is that gradient based optimisation approaches starting with random initialisations may end up in poor local minima [Bengio et al., 2006]. In 2006 unsupervised greedy layer-wise pre-training was proposed by [Hinton et al., 2006] to address the difficulty of training multi-layer neural networks. Later [Bengio et al., 2006] also concluded that this pre-training approach better optimises the parameters of a multi-layer neural network. In this approach, in each layer feature detectors in the neural network are first initialised using a stack of generative models. Restricted Boltzman Machines (RBMs) are used as generative models. The RBM has one layer of latent variables and is trained without having information of the target labels that a neural network needs to discriminate. In the second step, each layer of the neural network is initialised one layer at a time with a stack of trained RBMs assigned to each layer. Finally the parameters of the neural network are trained discriminatively using gradient optimisation techniques (like stochastic gradient descent) and target labels. The proposed pre-training technique has shown significant improvements in ASR WERs [Hinton et al., 2012; Dahl et al., 2012; Mohamed et al., 2012, 2009] and in image classification accuracy [Krizhevsky et al., 2012]. In [Glorot and...
an attempt was made to find what makes training deep neural networks with random initialisation difficult. A new initialisation scheme was proposed, based on experimenting with different activation functions, observing gradients and their flow across layers. In [Sutskever et al., 2013] the importance of initialisation by scaling the input to hidden connections is also highlighted. Given the importance of initialisation in this work we propose a pre-training algorithm for recurrent neural network language models.

### 4.3 Pre-training for RNNLMs

The proposed pre-training algorithm consists of three phases. Pre-training for RNNLMs (PT-RNNLMs) is accomplished by sharing the output layer weights of both NNLM and RNNLM. In the first phase of training, the weights of the NNLM are trained using the back propagation algorithm (described in Section 2.5.2). The NNLM is shown in Figure 4.1. The shared output layer weights are shown in red.

Figure 4.1: Feed-forward neural network language model. Here the output layer weights are shared with output layer weights of the RNNLM (shown in red).
In the second phase of the training the output layer weights of the RNNLM are shared and initialised (shown in red in Figure 4.2) with the output layer weights of NNLM (shown in red in Figure 4.1).

![Figure 4.2: Recurrent neural network language model. Here the output layer weights are initialised with the output layer weights of NNLM](image)

In the final phase of the training, random initialised weights \((U, V)\) and pre-trained weights \((W)\) are fine-tuned on the entire training data.

## 4.4 Perplexity Experiments

We have conducted perplexity (PPL) based experiments to evaluate the various hypotheses described in Section 4.1.

### 4.4.1 Prediction accuracy

We report performed perplexity experiments on the Wall Street Journal (WSJ) data in the Penn Treebank (PTB)\(^1\). The description of the PTB dataset is given in Sec-

\(^1\)http://www.cis.upenn.edu/ treebank/
A 10,000 (10K) word vocabulary was used, with out-of-vocabulary words being mapped to a special token <unk>. The number of tokens in the training, validation, and test sets were 930K, 74K and 82K words respectively. In each experiment reported in this section, 100 hidden nodes were used in the NNLM, RNNLM and PT-RNNLM. To reduce the computational complexity, a factored output layer is used. The factored output layer has two softmax layers. The first softmax layer predicts the probability of the class label given the context. The second softmax layer only predicts the probability of the words in the target class. The words are grouped into classes according to their frequency of occurrence in the training data. More details about the factored output layer is given in Section 4.4.4. Predicting only the words in the target class significantly reduces the number of computations in the output layer, which improves the training and test speed. In the traditional approach we need to compute the probability of all the words, which significantly increases the number of computations in the output layer. The dimension of the features in NNLM is 50 (dimension of word embedding). The baseline is Kneser-Ney smoothed 3-gram (KN3) with default count cut-offs. The RNNLM and NNLM hidden layer consists of 100 neurons. RNNLM output layer consists of 100 classes. The models are trained by stochastic gradient descent using the back propagation algorithm (weights were updated after each training example). A learning rate of $0.1^2$ was used to train the NNLMs and RNNLMs. A learning rate scheduler was used to halve the learning rate and for early stop training, when there is no change in entropy on validation data in two successive iterations.

In the PT-RNNLM, first the weights of the NNLM were fine tuned (first step), until the difference in entropy between iterations on a validation set was smaller than a pre-defined threshold. The weights of the RNNLM were then fine tuned (third step), by initialising the output layer parameters with the fine tuned output layer parameters of NNLM (second step). Perplexity (PPL) results on validation and test data before and after interpolation with $n$-grams are given in Figure 4.3 and Figure 4.4 respectively.

We can observe 4.2% and 3.0% relative improvements in PPL for the RNNLM over the 3-gram baseline, on validation and test sets respectively (in Figure 4.3 and Figure 4.4). The PT-RNNLM further reduces PPL by 3.0% and 3.3% relative on the validation and test sets. There is a further 10–15% relative reduction in PPL (on both data sets) when the neural network models are interpolated (the interpolation coefficient is 0.5) with the KN3 baseline LMs.

2After several experiments it was concluded that gains with 0.1 better other learning rates
3We always interpolate the RNNLMs with $n$-grams, since they are complementary to each other
Chapter 4. Feed-forward Pre-training for Recurrent Neural Network Language Models

Figure 4.3: Perplexities of 3-gram, NNLM, RNNLM and PT-RNNLM on validation data of PTB. The blue and green bars represents the PPL before and after interpolation, respectively.

Figure 4.4: Perplexities of 3-gram, NNLM, RNNLM and PT-RNNLM on test data of PTB. The blue and green bars represents the PPL before and after interpolation, respectively.
4.4.2 How many pre-training iterations are required?

We conducted another perplexity experiment on PTB data to investigate prediction accuracy of the PT-RNNLM with respect to the number of iterations (weight updates) of pre-training. The plot depicting the variation of PPL on test data with respect to the number of pre-trained iterations is shown in Figure 4.5.

From Figure 4.5, we can observe that there is a significant reduction in PPL after just one iteration of pre-training, and a slight downward trend for a further eight iterations, before the curve flattens. From this we can conclude a few iterations of pre-training is necessary to improve the prediction accuracy of RNNLM, and the feed-forward network does not need to be trained to completion. The same trend can be observed on validation data also, as depicted in Figure 4.6.

![Figure 4.5: The variation of PPL with respect to the number of pre-trained iterations (on test data). The same configuration of RNNLM and PT-RNNLM is used as described in Section 4.4.1.](image)

4.4.3 Number of iterations to converge

We have carried out an experiment to investigate whether the PT-RNNLM converges faster than the RNNLMs. The plot depicting the number of iterations required to converge to a local minimum is shown in Figure 4.7 for both RNNLM and PT-RNNLM. From the Figure 4.7, we can observe that pre-trained RNNLM took fewer iterations to
converge to a local minimum than the RNNLM. So we can conclude that pre-trained RNNLMs converge (fewer number of iterations) faster than the RNNLMs.

4.4.4 Factored Output Layer

To reduce the number of computations we used factorisation in the output layer. In factorisation, words will be grouped into classes according to the frequency of their occurrence or syntactic or semantic similarities. During training, first the distribution of the classes is computed given the context. Second only the distributions of words in that class will be computed. In factorisation, the probability of a word given the context is:

\[ p(w_i|w_{i-1}, h(t-1)) = p(c_i|h(t))p(w_i|c_i, h(t)) \]  (4.1)

where \( c_i \) class label assigned to a word \( w_i \).

Without factorisation the number of computations in the output layer is:

\[ H \times N \]  (4.2)

where \( H \) and \( N \) are the number of hidden neurons and size of vocabulary, respectively. Usually here \( H << N \).
After factorisation the number computations in the output layer would be:

\[ H \times (C + N_{c_i}) \]  

(4.3)

where, \( C \) and \( N_{c_i} \) are number of classes and number of words in a class, respectively. The \( N_{c_i} \) will vary according the number of classes and the type of algorithm used to group the words into classes. Here \( (C + N_{c_i}) \ll N \), this significantly reduces the number of computations in the output layer.

In the current work we assign the class label to the words according the frequency of their occurrence in the training data. For example if the number classes are 100, words covering the first 1% of unigram probability mass will be assigned to class label 1 and the words covering the second 1% of unigram probability mass will be assigned to class label 2 and so on. This is also called frequency binning.
4.5 ASR Experiments

The proposed RNNLMs are incorporated into ASR by rescoring the $N$–best lists (in all the experiments reported here $N=100$) of each utterance. The experimental setup to rescore the $N$–best lists is described in Section 3.3. Details about the ASR task (TED), acoustic and language models are described in the following subsections.

4.5.1 TED ASR task

The task we chose to evaluate the pre-trained RNNLMs was the TED transcription task, described in Section 3.2.2. The reasons we chose this task to evaluate the proposed RNNLMs are two fold. First, the audio quality of TED talks is very good, since speakers typically use a head-mounted microphone. Second, the task is challenging given the fact that vocabulary is very diverse (from different topics) and the lectures cover diverse topics.

In this work, we used the dev2010 and tst2010 sets for development and the tst2011 set for evaluation. In ASR experiments we always interpolate the scores of RNNLM with the $n$–gram scores, since $n$–gram and RNNLM are complementary to each other. The interpolation coefficients between the $n$–grams and the RNNLMs are optimised on development data (combination of dev2010 and tst2010) for best WERs.

4.5.2 Acoustic Models

The ASR system uses deep neural networks (DNNs) for training the acoustic models. The DNNs are trained in both Tandem and Hybrid configurations.

In Tandem systems the output classes of DNN are monophones [Bell et al., 2013b]. Once the DNNs are fine-tuned on training data, the output posteriors are decorrelated and reduced to low-dimensional (30 to 50) PCA features. The dimensionality reduced features were used to train HMM-GMM systems from scratch. The objective function of DNN is to optimise the frame classification rate on validation data, not the WERs.

In Hybrid setup, the DNN output classes are context-dependent phones [Bell et al., 2013b]. The likelihood estimates (which are equivalent to state distribution in HMM-GMM systems) are obtained by scaling the output posteriors using priors. These priors were estimated on the entire training data. The transition probabilities between the states are obtained from trained the HMM-GMM system. To compare Hybrid and Tandem systems: the context-dependent DNNs in Hybrid setup models the phones
with high resolution, whereas the Tandem system combines the benefits of DNNs and HMM-GMM systems.

In this work we use both types of system to evaluate the proposed pre-training for RNNLMs. Our goal is not to discover which setup (Hybrid or Tandem) will give more improvements, rather we would like to investigate the consistency of improvements across Hybrid and Tandem experimental configurations.

In addition to in-domain data, to take advantage of out-of-domain (OOD) data both the Tandem and Hybrid systems used Multi Level Adaptive Neural Network (MLAN) architecture for domain adaptation [Bell et al., 2013b; Bell et al.]. The Tandem and Hybrid acoustic models are trained on 143 hours of transcribed and aligned TED lectures, and 127 hours of transcribed AMI meeting data. Four hidden layer DNNs are used in Tandem setup, with each layer consisting of 1024 hidden neurons, whereas in the Hybrid setup the DNNs consists of six hidden layers each with 2048 hidden neurons.

### 4.5.3 Language Models

The following sections describe the $n$–gram and RNN language models used in the final ASR system.

#### 4.5.3.1 $n$–grams

The ASR system uses Kneser-Ney (KN) smoothed $n$–gram language models for decoding and lattice rescoring. As described in Section 3.2.2, the TED task provided us with 2.4M tokens of in-domain data (a collection of TED lectures) and large amounts of out-of-domain (OOD) data (3.6B tokens). Using the data selection process described in Section 3.2.2.2, a total of 312M tokens were selected from OOD data to train the $n$–gram LMs.

The $n$-gram language models are obtained by interpolating the language models trained on in-domain (2.4M) and selected subset of OOD data (312M). The vocabulary size of this task is 62,522 tokens. The interpolation coefficients were optimised on development data (combination of dev2010 and tst2010). In the final ASR system KN smoothed 3-gram and 4-gram LMs are used for decoding and lattice rescoring, respectively.

[https://www.idiap.ch/dataset/ami](https://www.idiap.ch/dataset/ami)
4.5.3.2 RNNLMs

RNNLMs were also trained on a combination of in-domain and OOD data. As mentioned above, a total of 312M tokens were selected from the OOD data. Given the complexity of training RNNLMs and PT-RNNLMs on large amounts of data, we trained the RNNLMs and PT-RNNLMs on a combination of in-domain and different smaller subsets of OOD data. Again we used a cross-entropy difference (CED) metric to create subsets of OOD data. A total of three OOD subsets were selected from the 312M OOD tokens, by controlling the CED metric threshold. The number of tokens after combining with the OOD data are given in Table 4.1. The vocabulary to train the RNNLMs and PT-RNNLMs consists of all the distinct words from in-domain and most frequent words in the OOD data. The remaining words in OOD data, which were not part of the vocabulary were replaced with a special token, \texttt{<unk>}.

<table>
<thead>
<tr>
<th>#Words</th>
<th>#Sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4M</td>
<td>749.3K</td>
</tr>
<tr>
<td>12.4M</td>
<td>1298.8K</td>
</tr>
<tr>
<td>22.4M</td>
<td>2011.3K</td>
</tr>
</tbody>
</table>

Table 4.1: Number of tokens after combining the in-domain data with subsets of OOD data. In-domain data consists of 2.4M tokens.

We used Mikolov’s RNNLM toolkit [Mikolov et al., 2011b] to train the RNNLMs and PT-RNNLMs. We followed the steps described in Section 4.3 to obtain PT-RNNLMs. Details of the NNLM are given in following subsection. To support initialisation of the output parameters of RNNLM with the NNLM, both the NNLM and RNNLM consist of the same number of hidden neurons. The neural nets are trained until the difference in entropy between two successive iterations is less than a pre-defined threshold. In all the experiments reported here we used a threshold of 0.01. The validation data for early stopping – the combination of dev2010 and tst2010 – contains a total 44K tokens.

4.5.3.3 NNLMs

NNLMs were trained to initialise the output layer parameters of RNNLM (first step in Section 4.3). 3-gram NNLMs were trained on the three datasets described in Table 4.1. The dimensionality of the projection layer was 50. The hidden layer consists
of the same number of neurons as RNNLM. Word classes were used in the output layer to reduce the computational complexity. A learning scheduler was used for early stopping, explained in Section 4.5.3.2. Even though a few iterations of pre-training is sufficient (from Figures 4.5 and 4.6), for the ASR experiments the NNLMs are fully trained.

### 4.5.4 Experimental Results

We report WERs on the dev2010, tst2010 and tst2011 data sets using Tandem MLAN HMMs and Hybrid MLAN HMMs described in Section 4.5.2. The RNNLMs and PT-RNNLMs are trained on combinations of in-domain data (2.4M word tokens) and three subsets of OOD data (5M, 10M and 20M tokens), as described in Table 4.1. The number of hidden neurons and the number of classes in the output layer are optimised for better WER. The models trained on 7.4M tokens consist of 300 hidden neurons and the models trained on 12.4M and 22.4M tokens consist of 500 hidden neurons. In all the experiments reported here 100 classes are used in the output layer to reduce the computational complexity. Compared to the RNNLM, it takes more time to train the PT-RNNLM since the weights of NNLM are fine tuned before fine tuning the weights of RNNLM. The initial learning rate for all the models was 0.1. We used a learning rate scheduler to halve the learning rate when the difference in entropy of two successive iterations is less than a pre-defined threshold (threshold was 0.01) and to early-stop the training.

We carried out WER experiments to investigate two questions. First, whether the PPL improvements (in Section 4.4.1) are transferable to WER improvements or not. Second, what are the improvements with respect to amount of OOD data used to train the models? We have also conducted tests to check whether the improvements are statistically significant or not.

#### 4.5.4.1 Tandem MLAN HMMs

The WERs after rescoring the 100-best lists of Tandem MLAN HMMs are given in Figure 4.8. Figures 4.8a, 4.8b and 4.8c depict the WERs of models which were trained on 7.4M, 12.4M and 22.4M tokens, respectively. In Figure 4.8a we can observe that RNNLM improves the 4-gram baseline on all the datasets and the percent absolute improvements are in the range of 0.1% to 0.4%. Similarly in Figure 4.8b and Figure 4.8c. In the same Figure 4.8a we can also observe that proposed PT-RNNLM reduces the
WERs by 0.1% over the RNNLM. The models trained on more OOD data in combination with the in-domain data further reduce the WERs. In Figure 4.8b, the reductions in the WERs with the PT-RNNLM are in the range of 0.06%-0.1%. Surprisingly, the WER of PT-RNNLM on tst2011 is slightly higher than RNNLM. Similarly, the models trained on 22.4M tokens further reduce the WERs, in Figure 4.8c. If we compare all the reported results on development and test sets, the proposed PT-RNNLM reduces the WERs in most of the cases and the reductions are consistent across data sets.

### 4.5.4.2 Hybrid MLAN HMMs

We also report the WER after rescoring the 100-best lists produced using Hybrid MLAN HMMs. In Figure 4.9a, we can observe, the PT-RNNLM reduces WERs of dev2010 by 0.1%, tst2011 by 0.25% and the WER of tst2010 is almost equal to that of RNNLM. As expected, the models trained on more OOD data in combination with the in-domain data further reduce the WERs, in Figure 4.9b and Figure 4.9c. In Figure 4.9b and Figure 4.9c, we can observe the reductions in WERs are in the range of 0.1%-0.23% and 0.1%-0.15%, respectively. Overall on average the proposed PT-RNNLM reduces the WERs by 0.15% absolute.

### 4.5.4.3 Statistical Significance Testing

We have conducted statistical significance testing to investigate whether the improvements are statistically significant. To measure the statistical significance between two speech recognition algorithms, a paired test can be run on their respective error rates, measured per utterance or per segment so as to represent a set of independent samples [Gillick and Cox, 1989]. In the present case, the appropriate granularity of measurement was judged to be at the level of results per TED talk, since the results within any one talk were on a fixed speaker and topic, and hence were not independent samples, but the speaker and topic changed between talks. Also it was the overall difference between RNNLM and PT-RNNLM that was of interest, rather than the difference on a specific training set size and with a specific acoustic model type. Therefore, a paired samples t-test (two-tailed) was run on the per-talk average word error rates with RNNLM and PT-RNNLM, where the averaging for each talk and language modelling technique was over the three training data sizes (7.4M, 12.4M and 22.4M words) and the Tandem and Hybrid models. This was done on the full set of 27 talks (8 in dev2010, 11 in tst2010 and 8 in tst2011). The result was a highly significant difference ($p = 0.000275$) in
4.5. ASR Experiments

Figure 4.8: WERs of dev2010, tst2010 and tst2011 using Tandem MLAN HMM system. (a) Here the models were trained on 5M tokens of OOD and 2.4M tokens of in-domain data. (b) WERs of models which were trained on a total 12.4M tokens. (c) A total of 22.4M tokens were used to train the models, out of which 20M tokens were OOD data.
Figure 4.9: WERs of dev2010, tst2010 and tst2011 using Hybrid MLAN HMM system. (a) Here the models were trained on 5M tokens of OOD and 2.4M tokens of in-domain data. (b) WERs of models which were trained on a total 12.4M tokens (C) A total of 22.4M tokens were used to train the models, out of which 20M tokens were OOD data.
favour of PT-RNNLM. This is significant at the 0.001 level, as recommended in a recent analysis of significance levels with regard to strength of evidence assessed by the corresponding Bayesian tests [Johnson 2013].

4.6 Discussion

From the experimental results depicted in Figure 4.8 and Figure 4.9, we can conclude that the PPL improvements are transferable to WERs, the percent improvements increase with respect to the amount of data used to train the models and improvements are consistent across datasets.

There are a number of further approaches to combine feed-forward and recurrent NNLMs. This includes co-training rather than pre-training (i.e. training the output weights is interleaved between the feed-forward and recurrent neural networks), sharing the projected continuous space representations between the two architectures and combining the feed-forward and recurrent architectures without output weight sharing (with one softmax operation).

4.7 Summary and Conclusions

In this work we have proposed feed forward pre-training for RNNLMs by sharing the output weights of the NNLM with the RNNLM. We report perplexity results on PTB data (in Section 4.4.1), with the proposed PT-RNNLM reducing perplexity on both the validation and test sets. We also investigated how many iterations of feed forward pre-training is necessary to improve the prediction accuracy of RNNLM. We have observed a few iterations of pre-training is sufficient. We also found that PT-RNNLMs converge faster (takes fewer iterations) than the RNNLMs. Finally we report the WERs by rescoring 100-best lists of Tandem and Hybrid MLAN HMMs for the IWSLT TED talk speech recognition task. We observed small but consistent gains across datasets. We also found that the WER gains are statistical significant with a $p$–value of $<0.001$. 


Chapter 5

Unsupervised Adaptation of Recurrent Neural Network Language Models

In this chapter we investigate supervised and unsupervised discriminative adaptation of RNNLMs in a broadcast transcription task to target domains defined by a show. We investigated two approaches based on (1) scaling forward-propagated hidden activations (The Learning Hidden Unit Contributions (LHUC) technique) and (2) adaptation of all the weights of RNNLM. To investigate the effectiveness of the proposed methods we carry out experiments on multi-genre broadcast (MGB) data following the MGB-2015 challenge protocol. We observe small but significant improvements in WER compared to a strong unadapted RNNLM model.

5.1 Introduction

The main goal of this chapter is to investigate the adaptation of RNNLMs to a target domain at test time using first pass recognition transcripts. We call this adaptation unsupervised adaptation. Why have we chosen RNNLMs for the adaptation? Adapting parametrised models like NNLMs and RNNLMs is easier and more flexible than adapting the $n$–grams ($n$–grams are non-parametrised in the sense that probability is estimated from the frequency of occurrences of words). Also neural network based models are the current state-of-the-art in language modelling.

Most large vocabulary continuous speech recognition (LVCSR) systems use broad
coverage LMs trained on millions or billions of tokens [Chelba et al., 2013]. These LMs are trained on text data that usually includes different domains, topics or genres. However, if the language model is more precisely matched to the test conditions (in terms of domain, topic, or data source) then improvements in WER might be expected. However, access to large amounts of in-domain data during LM training is usually limited, since transcribing audio data by humans is quite expensive, time consuming and for some domains we may not have enough data. To match the test conditions, a background LM is often first trained on a large amount of out-of-domain (OOD) text and then interpolated with a smaller in-domain LM. The interpolation coefficients are optimised on heldout data. However, these approaches still rely on identifying a sub-corpus of in-domain material: alternatively, LM adaptation can be carried out explicitly using unsupervised approaches to adapt the language model to the test data.

In this work, we have performed ASR experiments to answer three questions. First, will unsupervised adaptation improve the WERs? Second, are the gains for unsupervised adaptation better than for supervised adaptation? Finally, is there any correlation between the amount of training data used to train the models and the improvements after unsupervised adaptation? At the end we also discuss potential difficulties of adapting RNRLMs using first pass recognition transcripts.

The rest of the chapter is organised as follows. In Section 5.2, a description of previous attempts for language model adaptation is given. The proposed research problem is described in Section 5.3. Sections 5.4, 5.5 and 5.6 present experimental setup, results and analysis. Finally, a summary and conclusions are given in Section 5.7.

### 5.2 Previous Work

Adaptation for neural network-based language models falls into three categories: feature based adaptation, model based adaptation and discriminative methods for adaptation.

In feature based adaptation, input word features are augmented with domain specific features or an extra feature layer is added to the neural network to adapt to a domain or topic or both, during training and testing. Chen et al. [2015a] explored the explicit adaptation of RNRLMs to genre and topic, through fine-tuning on in-domain data (genre specific), the use of a metadata-derived genre code as an additional input feature, as well as the automatic extraction of topic representations (computed using

---

3 Adapted on human transcribed domain specific data by interpolation
latent Dirichlet allocation [Blei et al., 2003], probabilistic latent semantic analysis or hierarchical Dirichlet process modelling) as an additional input feature. This work was carried out on multi-genre broadcast (MGB) data used in the MGB challenge [Bell et al., 2015] and experiments were carried out at both the genre level and show level, with show-level adaptation consistently out-performing genre-level adaptation. For factored RNNLMs, the RNNLMs are provided with some structural information by appending structural feature vectors (POS, Lemma and Stem) to the input feature vectors [Wu et al., 2012]. In context-dependent RNNLMs [Mikolov and Zweig, 2012], context is enhanced by providing the RNNLM with topic proportions computed from a fixed number of words preceding the current word. The context-enhanced RNNLMs have shown significant WER reductions on the WSJ speech recognition task.

In standard maximum likelihood estimation (MLE) training for acoustic models, the objective is to optimise the likelihood of acoustic vectors given the transcripts. The MLE objective doesn’t consider other possible hypotheses during training. In contrast, discriminative training considers other possible hypotheses and reduces the probability of incorrect hypotheses or in other words reduces the number of errors. Discriminative training for GMM and DNN-based acoustic models significantly improves the WERs of various ASR tasks [Povey et al., 2008; Veselý et al., 2013; Woodland and Povey, 2002]. Similar to the acoustic models, language models are also trained discriminatively using all possible hypotheses. In discriminative language models (DLM) counts from both reference and ASR hypotheses (N-best lists or lattices) are used to correct the errors [Roark et al., 2004; Dikici et al., 2013]. In recent work, discriminative training is also extended to RNNLMs [Tachioka and Watanabe, 2015] where the objective is to reduce the cross-entropy between the target and output probability distributions. Similar to MLE training, CE training doesn’t consider the competing hypotheses. We can view discriminative training as adaptation during test time using competing hypotheses. But discriminative training is computationally expensive and careful tuning of various parameters is required. In this work, we proposed an adaptation framework during test time, with fewer computations required.

In model based adaptation, all or a subset of the parameters of the model are updated to adapt the model. A model can also be adapted by adding linear or non-linear layers between the existing layers in a network. In Multi-Domain RNNLMs [Tilk and Alumåe, 2014], a bottleneck (or compression) layer is added between the hidden and the output layer, which is then estimated on adaptation data. A domain feature vector is connected to the newly added compression layer, where each dimension in the fea-
ture vector represents one domain. A single RNNLM is trained to adapt to multiple domains. Also, some work has been done on domain adaptation using feed-forward NNLMs by adding an adaptation linear layer between the projection and hidden layers [Park et al., 2010].

In other adaptation approaches, data similar to the target domain was selected based on a cross-entropy difference metric and models were trained on a combination of in-domain and selected data [Moore and Lewis, 2010; Bell et al., 2013a; Gangireddy et al., 2014]. A maximum entropy framework is used to adapt the LMs to a topic and to the syntactic structure of the sentence [Khudanpur and Wu, 2000]. LMs can be also adapted to a target domain using information retrieval methods [Chen et al., 2003]. In [Bacchiani and Roark, 2003], n-grams are adapted to a target domain by merging the counts from ASR transcriptions and language model data.

5.3 Unsupervised Adaptation of RNNLMs

In this work, we investigate unsupervised adaptation of RNNLMs to a specific television programme (show), performing experiments on the MGB challenge transcription task [Bell et al., 2015], described in Section 3.2.3. The MGB data, which consists of subtitled BBC television broadcasts, includes metadata that enables both the show and its genre to be identified (the genre information is provided by the BBC). In our experiments we have focused on the adaptation of the RNNLMs to a specific show only.

The unsupervised framework for adapting RNNLMs is shown in Figure 5.1. In the unsupervised adaptation framework the adaptation is done during test time (after first pass recognition). In the first step, a light weight or pruned language model is used to generate 1-best hypothesis for the adaptation. In the second step, RNNLMs are adapted using the 1-best generated in the first pass recognition. Finally, adapted RNNLMs are used to rescore the N-best lists OR lattices.

We investigate two RNNLM adaptation methods. The first one relies on learning show-dependent amplitudes of the hidden unit contributions (LHUC) [Swietojanski and Renals, 2014; Swietojanski et al., 2016]. The second approach directly updates the parameters of the background RNNLM, referred to as adapt-all-parameters. The proposed two methods are described in the following sections.
5.3. Unsupervised Adaptation of RNNLMs

5.3.1 Learning Hidden Unit Contributions (LHUC)

As described in Section 2.5.1 and Section 2.5.2, neural network based models learn a non-linear transfer function. For a given data \( x(t) \) the non-linear transformation is:

\[
 f(x_t; \theta) = \phi(W \phi^L(W^L \phi^{L-1}(... \phi^1(W^1 x(t))...))) \tag{5.1}
\]

where, \( \phi^l \) is the non-linear transform for layer \( l \) and \( \phi \) is output layer transform. The output of a hidden layer \( l \) can be written as:

\[
 h^l(t) = \phi^l(W^l h^{l-1}(t)) \tag{5.2}
\]

The model parameters are given by \( \theta = \{W^1, ..., W^L, W\} \). Generally \( \phi^l \) is sigmoid \((1/(1 + \exp(-c)))\) or tanh \(((1 - \exp(-c))/(1 + \exp(-c)))\) or rectified linear units (ReLU) or the maxout function. A neural network based language model like RNNLM contains one recurrent hidden layer, one projection layer and an output layer. For RNNLM, the non-linear transfer function in Equation 5.1 becomes:

\[
 h(t) = \phi^1(U x(t) + V h(t-1)) \tag{5.3}
\]
\[
 f(x(t), h(t-1); \theta) = \phi(W h(t)) \tag{5.4}
\]

where \( x(t) \) is the input at time \( t \), \( U \) is a matrix connecting input and hidden layer (word embeddings), \( h(t-1) \) is state of hidden layer at time \( t-1 \) and \( \phi^1 \) is non-linear transform for hidden layer. The model parameters are \( \theta = \{W, V, U\} \). The output activations represents the probability of a word given the input and the context.

\[
 P(w_t = k|w_{t-1}, h(t-1); \theta) = \frac{e^{W_k h(t)}}{\sum_{k=1}^K e^{W_k h(t)}} \tag{5.5}
\]
where \( K \) is the size of the output layer, equal to vocabulary size. The model is usually trained on language model training data which consists of data from a combination of various domains or topics. When this type of LM is tested on data from other domains or topics which are not part of the training data, the generalisation is expected to be poor. Adaptation of LMs during train or test time can improve the generalisation to sequences of words from unseen domains or topics. This can be achieved by using some adaptation data for each domain or topic, \( \{x^m(t), y^m(t)\} \), where \( m \) represents a domain or topic index, \( x^m \) and \( y^m \) are input and corresponding output label respectively.

In the current work the target domain is a show. The adaptation updates the model such that it better approximates the target probability distribution, \( P(w^m_i | w^m_{i-1}, h(t - 1); \theta^m, \theta) \). We modify the domain- or topic-independent distribution by defining a set of show specific parameters, \( \theta^m = \{r_m\} \), where \( r_m \in \mathbb{R}^M \) is the vector of show specific parameters. Using adaptation parameters \( r_m \) Equation 5.2 can be modified as:

\[
\begin{align*}
    h'(t) &= h(t) \circ a(r_m) \\
    a(c) &= \frac{2}{1 + e^{-c}}
\end{align*}
\]

where \( h(t)' \) are the hidden activations after scaling and \( \circ \) denotes element-wise multiplication. The re-parametrisation function (Eq. 5.7) is defined as a sigmoid with amplitude of 2.0, which gives an effective scaling range of \([0, 2]\). The sigmoid is applied to control the effective range of scaling parameters. The \( a(r_m) \) can be viewed as a scaling factors for hidden activations during test time. These \( r_m \) are estimated on adaptation data.

We refer to this approach as Learning Hidden Unit Contributions (LHUC) [Swietojanski et al., 2016]. It allows the hidden units to be re-weighted according to their relative importance in modelling the domain/topic/show specific distribution over sequences of words. After scaling the parameters the next layer receives the re-weighed activations - some activations are scaled up, some are scaled down and some are unchanged. The potential advantages of this method are as follows: (i) the number of adaptation parameters \( \theta_m \) are just 0.00001% of \( \theta \); and (ii) the scaling of hidden activations does not alter the learned feature detectors which is a desired property when adapting with small amounts of noisy adaptation targets. Also this method is robust against over-fitting.

The domain or topic or genre specific parameters are learned by optimising the cross-entropy (CE) between the target and output probability distributions:
5.3. Unsupervised Adaptation of RNNLMs

\[
CE = - \sum_{k=1}^{K} y_k^m \log p_k^m
\]  

(5.8)

where \(y_k^m\) and \(p_k^m\) are respectively the target and output probability distributions for a specific domain/topic/genre \(m\). During adaptation, only adaptation parameters \((r_m)\) are updated and the remaining parameters \((\theta)\) are not updated.

\[
r_m(t + 1) = r_m(t) - \alpha_{\text{adapt}} \frac{\partial(CE)}{\partial r_m}
\]  

(5.9)

where \(\alpha_{\text{adapt}}\) is adaptation learning rate.

\[
\frac{\partial(CE)}{\partial a(r_m)} = \frac{\partial(CE)}{\partial h'(t)} \frac{\partial h'(t)}{\partial a(r_m)}
\]  

(5.10)

where \(\frac{\partial a(r_m)}{\partial r_m}\) depends on the non-linear function used in the hidden layer. \(\frac{\partial(CE)}{\partial a(r_m)}\) is computed as follows.

\[
\frac{\partial(CE)}{\partial a(r_m)} = \sum_{k=1}^{K} \frac{\partial(CE)}{\partial o_k} w_{kj}
\]  

(5.12)

\[
= \sum_{k=1}^{K} (y_k^m - p_k^m) w_{kj}
\]  

(5.13)

\[
\frac{\partial h'(t)}{\partial a(r_m)} = a(r_m)(1 - a(r_m))
\]  

(5.14)

As described in Section 5.3, we learn one \(r_m\) for each show. So the target domain is a specific BBC show. In this work, we have also found it beneficial to update only forward-pass activations for adaptation, which are passed unscaled to the recurrent layer \(h'(t)\), in order to avoid modifying the learned history.
5.3.2 Adapt parameters of RNNLM

In model based adaptation, the other possibility would be directly adapt the parameters of RNNLM during test time using adaptation data. We refer to this approach as *adapt-all-parameters*. We can adapt all parameters or a subset of parameters of RNNLM. We use 1-best transcripts from first pass recognition to adapt the parameters of the RNNLM.

5.3.2.1 Adapt all Parameters (RNNLM-Adapt-All)

In this method we adapt all the parameters of RNNLM (trained on a large amount of language model training data) using show specific adaptation data (obtained from first pass decoding here). As described in Section 5.3.1, if $\theta$ are the parameters of RNNLM then the adapted parameters are $\theta_{m\text{adapt-all}}$ where $m$ is the index of BBC show.

$$
\theta_{m\text{adapt-all}} = \{U^m, V^m, W^m\}
$$

(5.15)

These parameters are learned using the standard back propagation through time (BPTT) algorithm (described in Section 2.5.2.1). Since we are adapting all the parameters on relatively small amounts of adaptation data there is a possibility that the RNNLM might over-fit. We experimentally searched for the optimal learning rates on the development set, and we report the numbers for both high and low learning rates. In Section 5.6 we discuss the over-fitting issue and recommendations to address this problem.

5.3.2.2 Adapt Subset of Parameters (RNNLM-Adapt-Subset)

Instead of adapting all the parameters, we can also adapt the model to a target domain by adapting subset of parameters. If $\theta = \{U,V,W\}$ are parameters of a RNNLM, adapting subset of parameters results in:

$$
\theta_{m\text{adapt-input}} = \{U^m, V, W\}
$$

(5.16)

$$
\theta_{m\text{adapt-recurrent}} = \{U, V^m, W\}
$$

(5.17)

$$
\theta_{m\text{adapt-output}} = \{U, V, W^m\}
$$

(5.18)
5.4 ASR Experiments

To investigate the effectiveness of RNNLMs and adapted RNNLMs we rescored $N$-best lists (in all the experiments here $N=100$) of each utterance. The $N$-best list rescor- ing setup is described in Section 3.3. The task we choose to evaluate the proposed LMs, acoustic models, resources for language models ($n$-grams and RNNLMs) and adapted RNNLMs are described in following subsections.

5.4.1 MGB Task

Multi-Genre Broadcast (MGB) task [Bell et al., 2015] was one of the official challenge tasks at ASRU 2015. In addition to audio recordings, the dataset also consists of millions of tokens of subtitles. Unlike other broadcast speech recognition tasks [Gales et al., 2006; Gauvain et al., 2002], MGB task consists of recordings from broad genres (drama, quiz, documentaries, comedy, sports, politics, food, travel). The data from broad genres makes this task more challenging than other broadcast speech recognition tasks. More details about the data selection procedure for the acoustic model, language model data and available metadata are described in Section 3.2.3.

5.4.2 Acoustic Models

As described in Section 3.2.3.2 two error rate measures (PMER/MER and AWD) were used to select data segments for acoustic model training. We selected a total of 640 hours audio (from 1600 hours) using AWD in the range of [0.2, 0.7] and a 40% threshold on PMER. GMMs were trained on filterbank+pitch features using the standard Kaldi recipe [Povey et al., 2011]. A six-layer DNN with 2048 units in each layer was used to compute the posterior probability of tied states obtained from the GMM acoustic models. A total of 9 frames([-4, +4]) of acoustic features was given as input to the DNN. The total number of tied states was 28K. After fine-tuning the parameters of the DNN, two iterations of sequence training were applied [Vesely et al., 2013]. The pronunciations for the words obtained by using the combilex lexicon of British English [Richmond et al., 2009, 2010]. Pronunciations for missing words are automatically generated using grapheme-to-phoneme conversion [Bisani and Ney, 2008].

4asru2015.org
5.4.3 Language Models

As described in Section 3.2.3.3, a total of 650M tokens of subtitle data is available for language model training. Out of 650M tokens, 10M tokens are acoustic transcripts and we use these only for supervised adaptation experiments, but not for unsupervised adaptation experiments. Before training the LMs the text data was normalised, with numbers converted to text-form and abbreviations converted to sequences of letters.

We used dev.full and eval.task1 to tune the parameters and to evaluate the models, respectively. The OOV rates of dev2010 and eval.task1 are 0.5% and 2.0%, respectively.

5.4.3.1 N–grams

The vocabulary consists of the 150K most frequent words from training data (640M tokens). The most frequent words were selected based on unigram probability distribution. A Kneser-Ney smoothed pruned 3-gram\(^5\) was used in first pass decoding to generate the lattices and N–best lists. The first pass recognition resulted in a WER of 32.6% on dev.full. After first pass we rescored the lattices with a full 3-gram which resulted in a perplexity of 175.39 and a WER of 31.0% on dev.full. We used the 1-best hypothesis from the first pass decoding for unsupervised RNNLM experiments.

5.4.3.2 RNNLMs

RNNLMs were also trained on 640M tokens of BBC subtitle text data. Due to computational complexity of training RNNLMs on a vocabulary of 150K words, we trained the RNNLMs by creating input and output short-lists (proposed by Schwenk [2007] to reduce the computational complexity) consisting of the most frequent words from the 150K vocabulary: in the current work the input and output short-list sizes were 64K and 30K respectively. Both input and output layers had an extra node to compute the probability of out-of-short-list words, represented as \(<\text{oos}>\). During PPL computation and N–best list rescoring the probability of \(<\text{oos}>\) node is distributed equally among all short-list words. The RNNLM is trained with a batch size of 256 and learning rate of \(2.06\times10^{-3}\) (per sample learning rate=7.8125 \(\times\) 10\(^{-3}\)). The parameters of RNNLM are trained by stochastic gradient decent (SGD) using BPTT algorithm. During training the gradient errors were propagated five time steps back in time (equal to five words).
to learn the long-distance dependencies. The objective function is the cross-entropy between the output and target probability distributions. The hidden layer consisted of 512 nodes. RNNLMs were trained on graphics processing units (GPU) using the Cambridge RNNLM toolkit [Chen et al., 2014 2016].

5.4.3.3 RNNLM-LHUC

In LHUC adaptation, as described in Section 5.3.1 the scaling parameters were estimated on a first pass 1-best decoding transcripts (generated using a pruned 3-gram LM). During adaptation only the LHUC scaling parameters were updated and all remaining parameters are kept unchanged. Since the RNNLM is adapted to a show, one set of scaling parameters were estimated for each show in the dev.full and eval.task1 test sets. In addition to adapting the RNNLMs on the 1-best transcripts we also conducted oracle experiments using the reference transcripts. In oracle experiments we adapted the RNNLMs using reference transcripts of each show. We have done these experiments to investigate the lower bounds on WERs with this method. A learning rate of 1.0 (per sample learning rate=3.9 × 10^{-4}) was used to learn the LHUC parameters. Empirically we found that a learning rate of 1.0 is optimal with respect PPL on validation data. The show-specific adaptation parameters were reused during N–best list rescoring.

5.4.3.4 RNNLM-Adapt-All

In this method, all the parameters of RNNLM were adapted on first pass 1-best transcripts. The parameters were adapted by SGD using standard BPTT algorithm. Since we are adapting the RNNLMs on small amounts of adaptation data this method is not robust against over-fitting. We performed a number of experiments by varying the learning rate during adaptation. We report results using learning rates of 0.1 and 1.0. We also report adaptation experimental results by adapting a subset of the parameters of the RNNLM, as described in Section 5.3.2. We have done oracle experiments to investigate the lower bounds on WERs. Oracle experiments were done by adapting the parameters on reference transcripts of dev and eval test sets.
5.5 Experimental Results

The RNNLMs and adapted RNNLMs were incorporated into ASR by rescoring 100-best lists of \texttt{dev.full} and \texttt{eval.task1} transcription test sets of the MGB Challenge. The RNNLM scores were interpolated with the 3-gram scores and the interpolation coefficient was optimised for better WER on \texttt{dev.full}. To investigate the amount of data used to train the RNNLMs and gains after adaptation, we adapted the RNNLMs trained on 640M (RNNLM-640M) and on 40M (RNNLM-40M) tokens. The interpolation coefficient for RNNLM-640M experiments was 0.5 and for RNNLM-40M was 0.3.

5.5.1 RNNLM-LHUC

The WERs on \texttt{dev.full} and \texttt{eval.task1} using LHUC adaptation are given in Table 5.1. In the first row of the Table 5.1, the WERs using a pruned 3-gram LM are given. After rescoring with the full 3-gram LM we can observe 1.6\% and 1.4\% absolute improvements on \texttt{dev.full} and \texttt{eval.task1}, respectively. The WERs of the RNNLM trained on 640M words are given in the third row of Table 5.1, 3-gram+RNNLM-640M. With RNNLM-640M we can observe 0.7\% absolute improvements on both \texttt{dev.full} and \texttt{eval.task1}. LHUC adaptation improves the interpolated 3-gram and RNNLMs by 0.1\% absolute, on both \texttt{dev.full} and \texttt{eval.task1}. To find the lower bounds on WERs with the proposed LHUC method, we report the WERs by adapting the RNNLMs on reference transcripts of \texttt{dev.full} and \texttt{eval.task1}. When the RNNLM-640M is adapted on reference transcripts (3-gram+RNNLM-640M-lhuc-oracle) we can observe 0.2\% absolute improvements both on \texttt{dev.full} and \texttt{eval.task1}. From the experimental results we can conclude that LHUC adaptation improves the strong unadapted baseline by 0.1\% and the improvements are consistent across datasets.

To investigate the correlation between amount of data used to train the models and % improvements after adaptation, we have done LHUC adaptation experiments using a smaller training dataset, consisting of a total of 40M tokens. This dataset was formed by sampling from the 640M tokens dataset\footnote{Sampling was done taking first 15 utterances from contiguous blocks of 200 utterances in 640M dataset}. The WERs with models trained on 40M tokens are given in Table 5.2. The 3-gram baselines are the same as earlier experiments. With an RNNLM trained on 40M tokens we can observe 0.1\% and 0.2\% absolute improvements over the full 3-gram on \texttt{dev.full} and \texttt{eval.task1}, respectively. After LHUC adaptation, in the fourth row of Table 5.2 we can observe 0.1\% absolute
### 5.5. Experimental Results

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
<th>eval.task1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram-pruned</td>
<td>32.6</td>
<td>33.6</td>
</tr>
<tr>
<td>3-gram-rescored</td>
<td>31.0</td>
<td>32.2</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M</td>
<td>30.3</td>
<td>31.5</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M-1best</td>
<td>30.2</td>
<td>31.4</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M-oracle</td>
<td>30.1</td>
<td>31.3</td>
</tr>
</tbody>
</table>

Table 5.1: % WERs of RNNLM and adapted RNNLM by LHUC method. The RNNLMS were trained on 640M tokens. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used.

We have also done the oracle experiments to investigate the lower bounds on WERs with this data set. From the last row of the table we can observe that the lower bound on WER is 0.1% less than the unadapted baseline. From the experimental results we can conclude that there is no correlation between amount of data and % improvements.

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
<th>eval.task1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram-pruned</td>
<td>32.6</td>
<td>33.6</td>
</tr>
<tr>
<td>3-gram-rescored</td>
<td>31.0</td>
<td>32.2</td>
</tr>
<tr>
<td>3-gram+RNNLM-40M</td>
<td>30.9</td>
<td>32.0</td>
</tr>
<tr>
<td>3-gram+RNNLM-40M-1best</td>
<td>30.8</td>
<td>32.0</td>
</tr>
<tr>
<td>3-gram+RNNLM-40M-oracle</td>
<td>30.8</td>
<td>31.9</td>
</tr>
</tbody>
</table>

Table 5.2: % WERs of RNNLM and adapted RNNLM by LHUC method. The RNNLMS are trained on 40M tokens. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used.

#### 5.5.2 RNNLM-Adapt-All

In Table 5.3, we report the WERs on dev.full and eval.task1 by adapting all the parameters of RNNLM-640M. The 3-gram and RNNLM baselines are the same as above. By adapting all parameters of an RNNLM trained on 640M tokens, we can observe 0.1% absolute improvements on both dev.full and eval.task1. To investigate the lower bounds on WERs, we report the WERs by adapting all the parameters of RNNLMS on
reference transcripts of dev.full and eval.task1. From the fifth row of Table 5.3, we can observe that lower bounds are 0.3% and 0.4% less than the RNNLM baseline, on dev.full and eval.task1, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
<th>eval.task1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram-pruned</td>
<td>32.6</td>
<td>33.6</td>
</tr>
<tr>
<td>3-gram-rescored</td>
<td>31.0</td>
<td>32.2</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M</td>
<td>30.3</td>
<td>31.5</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M-1best</td>
<td>30.2</td>
<td>31.4</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M-oracle</td>
<td>29.9</td>
<td>31.1</td>
</tr>
</tbody>
</table>

Table 5.3: % WERs of RNNLM and adapted RNNLM by adapting all parameters. During adaptation a learning rate of 0.1 is used. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used.

We also report the WERs using models which were trained on the smaller training set, consisting of 40M tokens. We can observe 0.2% absolute improvements on both datasets using RNNLM-40M, in the third row of Table 5.4. After adaptation, the improvements are 0.1% absolute on both dev.full and eval.task1. To investigate the lower bounds on WERs we adapted the models on reference transcripts. The lower bounds are 0.2% less than the strong unadapted baseline (in the fifth row of Table 5.4).

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
<th>eval.task1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram-pruned</td>
<td>32.6</td>
<td>33.6</td>
</tr>
<tr>
<td>3-gram-rescored</td>
<td>31.0</td>
<td>32.2</td>
</tr>
<tr>
<td>3-gram+RNNLM-40M</td>
<td>30.9</td>
<td>32.0</td>
</tr>
<tr>
<td>3-gram+RNNLM-40M-1best</td>
<td>30.8</td>
<td>32.0</td>
</tr>
<tr>
<td>3-gram+RNNLM-40M-oracle</td>
<td>30.7</td>
<td>31.8</td>
</tr>
</tbody>
</table>

Table 5.4: % WERs of RNNLM and adapted RNNLM by adapting all the parameters. During adaptation a learning rate of 0.1 is used. For adaptation, the 1-best decoding and the reference transcripts of dev.full and eval.task1 are used.

Finally, we also did experiments by adapting a subset of parameters of RNNLM. In Table 5.5 we report the WERs on dev2010 by adapting the weights of one layer each time. Adapting the weights connecting the input and hidden layer is not effective enough to improve the baseline, as seen in the second and third rows of the Table 5.5. In
the fourth and fifth rows of the Table 5.5 we can observe 0.1% absolute improvement after adapting the hidden to hidden connections using 1-best and 0.2% absolute using reference transcripts, respectively. Similar to adapting input to hidden connections, adapting only hidden to output connections is also not effective enough to improve over the baseline, as seen in the sixth and seventh row of the Table 5.5. Here we used a learning rate of 0.1 and mini-batch size of 256.

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram+RNNLM640M</td>
<td>30.3</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer0-1best</td>
<td>30.3</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer0-oracle</td>
<td>30.3</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer1-1best</td>
<td>30.2</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer1-oracle</td>
<td>30.1</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer2-1best</td>
<td>30.3</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer2-oracle</td>
<td>30.3</td>
</tr>
</tbody>
</table>

Table 5.5: % WERs of RNNLM and adapted RNNLM by adapting subset of parameters. Adaptation learning rate is 0.1

In the Table 5.6 we report the WERs on dev.full achieved by adapting subset of parameters of RNNLM using an adaptation learning rate of 1.0. We can observe that, using a high learning rate over-fits the adaptation data and the WERs are higher than the baseline, in the fifth row of the Table 5.6. Similarly, adaptation over-fits the reference transcripts, as shown in sixth and eighth rows of the Table 5.6.

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram+RNNLM640M</td>
<td>30.3</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer0-1best</td>
<td>30.3</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer0-oracle</td>
<td>30.2</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer1-1best</td>
<td>30.4</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer1-oracle</td>
<td>29.5</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer2-1best</td>
<td>30.2</td>
</tr>
<tr>
<td>3-gram+RNNLM640M-layer2-oracle</td>
<td>29.7</td>
</tr>
</tbody>
</table>

Table 5.6: % WERs of RNNLM and adapted RNNLM by adapting subset of parameters. Adaptation learning rate is 1.0
From Table 5.5 and Table 5.6, we can conclude that adapting hidden to hidden connections improves over RNNLM baseline.

5.6 Discussion

Tables 5.1 and 5.3 show that both LHUC and adapt-all-parameter methods improve the WER by 0.1% absolute (0.3% relative) for RNNLMs trained on 640M tokens. The improvements are small but consistent across test sets. To find the statistical significance of improvements, we performed matched pair sentence segment word error (MPSSWE) [Gillick and Cox, 1989] tests for the proposed adaptation methods and baselines, for the RNNLM-640M case. The statistical significance test reveals the reported improvements, though small, are significant at \( p < 0.001 \) level. This is due to the fact that both test sets are relatively large – dev.full consists of 200K tokens or 28 hours of speech and eval.task1 consists of 80K tokens or 11 hours of speech.

As discussed in Section 5.3.1, LHUC is robust against over-fitting, since there are far fewer adaptation parameters than the total number of parameters in the RNN, and because feature receptors are not modified. This is not the case with adapt-all-parameters, in which all the parameters of RNN are altered based on small amounts of adaptation data. It is thus likely that the RNNLM can over-fit the adaptation data when adapting all the parameters. For the results reported in Table 5.3, we used a small learning rate of 0.1 (per sample learning rate=3.9 \( \times 10^{-4} \)), during adaptation. To investigate the effect of learning rate, we adapted the RNNLM trained on 640M tokens to a target show with a learning rate of 1.0 (per sample learning rate=3.9 \( \times 10^{-3} \)). In the first row of the Table 5.7, we can observe 1.4% absolute improvement on dev.full, by adapting the RNNLM on reference transcripts. In the second row of Table 5.7, we can observe that adaptation on the 1-best transcripts with a high learning rate has a higher WER than the baseline. In Table 5.7 we can also observe the PPLs before and after adaptation. After adaptation we can observe 72.3% and 59.4% relative improvements over the baseline on reference and 1-best transcripts, respectively. The improvements on the reference transcripts suggest that lower WERs in the unsupervised adaptation setting may be obtained once the adaptation process is properly regularised.

In Table 5.1 and Table 5.3, we report the average WERs of all the shows in dev.full and eval.task1. Given we adapted the RNNLMs at show level, we also looked at the WER (%) improvements at each show level. After adaptation, both proposed methods improve the WERs of almost all the shows, with fewer than 5 (out of 47) shows with
### 5.6. Discussion

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PPL</td>
<td>WER</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M</td>
<td>153.12</td>
<td>28.9</td>
</tr>
<tr>
<td>3-gram+RNNLM-640M</td>
<td>202.43</td>
<td>30.6</td>
</tr>
</tbody>
</table>

Table 5.7: % WERs baseline RNNLM and adapted RNNLM by Adapt-all method. With a learning rate of 1.0 increased WERs after adaptation. The WERs at show level are given in Table A.1 and Table A.2.

In the MGB Challenge transcription task we also have access to about 10M tokens from the transcriptions of the acoustic training data. Table 5.8 reports WERs on dev.full and eval.full obtained by adapting the RNNLMs on this data. Both the 3-gram and RNNLM are adapted using linear interpolation. The full 3-gram LM is interpolated with a 3-gram trained on the acoustic training transcripts data, with an interpolation coefficient of 0.9. The RNNLM-640M is interpolated with a RNNLM trained on the acoustic training transcripts with an interpolation coefficient of 0.9. From Table 5.8 we can observe that supervised adaptation improves the baseline by 0.1% absolute. This is a similar improvement to unsupervised adaptation on the test data reported above.

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
<th>eval.task1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram+RNNLM-640M</td>
<td>30.3</td>
<td>31.5</td>
</tr>
<tr>
<td>3-gram-adapt+RNNLM-640M-adapt</td>
<td>30.2</td>
<td>31.4</td>
</tr>
</tbody>
</table>

Table 5.8: % WERs of RNNLM and RNNLM adapted on 10M tokens of acoustic transcripts

To find the maximum gains that can be obtained by rescoring the 100-best lists, we report the 100-best oracle\(^8\) for dev.full in Table 5.9. We used Kaldi based tool (lattice-oracle) to compute this WER.

---

\(^8\)This WER was computed by converting the 100-best lists of each utterance to a lattice. 100-best lists are generated from the first pass decoding lattices.
Table 5.9: 100-best oracle for dev.full

<table>
<thead>
<tr>
<th>Model</th>
<th>dev.full</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-best-oracle</td>
<td>25.5</td>
</tr>
</tbody>
</table>

5.7 Summary

We have investigated unsupervised adaptation of RNNLMs to a target domain (BBC show in multi-genre broadcast transcription task), following the MGB Challenge protocol. We have investigated two adaptation scenarios – LHUC and fine-tuning. Our experimental results indicate that WER reductions arising from unsupervised test-only adaptation using either LHUC or fine-tuning are small but statistically significant.
In this chapter we propose to enhance the RNNLMs with prosodic and syntactic features computed using the context of the current word. First we enhanced the context with prosody features. Since it is plausible to compute the prosody features at the word and syllable level we have trained the models on prosody features computed at both these levels. To investigate the effectiveness of the proposed models we report perplexity and WER for two speech recognition tasks, Switchboard and TED. We observed substantial improvements in perplexity and small improvements in WER. In addition to prosody features, we also enhanced the context with syntactic features computed from the context of the current word. The syntactic features are part-of-speech (POS) tags and Combinatory Categorial Grammar (CCG) supertags [Steedman, 2000]. To investigate the effectiveness of these models we report the PPL and WERs on TED speech recognition task. We observed small but significant and consistent improvements across the datasets.

6.1 Introduction

The main goal of this chapter is to report our experiments into the enhancement of the RNNLMs with contextual features computed from the context of the current word. We accomplished this enhancement by adding an extra feature layer to the RNNLM. Why would we like to add a feature layer to RNNLM? Adding a feature layer to RNNLM provides a way to incorporate various contextual features, such as syntactic, semantic and acoustic prosody features. Adding a layer also allows us to incorporate other
external context information like location and features from user logs [Aleksic et al., Sep 2015]. What are the advantages of enhancing the context of a RNNLM? The advantages are two fold. First, adding contextual features can address the data sparsity issue (different contexts may have same features) where we have reduced amounts of in-domain data for automatic speech recognition. Second, contextual features can also improve generalisation to unseen sequences of words. The RNNLM with a feature layer \((F)\) is shown in Figure 6.1.

![Diagram of RNNLM with feature layer](image)

**Figure 6.1**: Recurrent neural network language model with a feature layer (green)

The activations at the hidden and output layer are computed as follows:
\[ h_t = f(Ux(t) + Vh(t-1) + Fc(t)) \]  \hspace{1cm} (6.1)
\[ y_t = g(Wh(t)), \]  \hspace{1cm} (6.2)

where \( f, g \) are sigmoid and softmax activation functions, respectively. Through the feature layer, the hidden layer can learn both word and feature context.

In the literature, a number of approaches have been proposed to incorporate long span and more contextual features into a LM. In cache LMs, additional context information is incorporated by interpolating the background LM with an LM trained on the last \( K \) words in the context [Kuhn and Mori, 1990]. In context dependent RNNLMs [Mikolov and Zweig, 2012], context is enhanced with topic proportions computed from the words in the most recent context. In [Khudanpur and Wu, 2000], maximum entropy LMs are trained on topic, semantic and syntactic features. In other works, LMs are trained by segmenting the training data. The segmented data consists of data related to a particular topic and at test time the most appropriate is chosen [Iyer and Ostendorf, 1996]. In a similar spirit to [Mikolov and Zweig, 2012], in this work we enhanced the context of RNNLM using prosody and syntactic features computed from the context of the current word.

In this chapter we have performed experiments to answer two different questions. First, will enhancing the context with prosody and syntactic features will improve the prediction accuracy or not? If prediction accuracy improves, will these improvements carry to speech recognition WERs? Second, will the prosody models work for less prosody-rich tasks like TED? At the end we also discuss experimental results of various prosody and syntactic features.

The rest of this chapter is organised as follows. In Section 6.2, we discuss the motivation for enhancing the context with prosody features and give experimental results. Motivation for syntactic features and experimental results are given in Section 6.3. In Section 6.4, we discuss the prosody and syntactic feature experimental results. Finally, a summary and conclusions are given in Section 6.5.

### 6.2 Context-enhancement Using Prosody Features

Current large vocabulary speech recognition systems typically comprise an acoustic model which relates acoustic features to sub-word units, a lexical model of some kind (typically a dictionary), and a language model which provides probability estimates...
for word sequences. Suprasegmental prosodic features, such as intonation and timing information, fit uneasily into such a framework. However, prosodic features are of potential interest in speech recognition: they are relatively robust to noise and provide rich additional information. Prosodic information is available at various levels in a speech signal: within words (for instance, word and phone duration), between the words (for instance, pause duration), and across multiple words (for instance, F0 contour).

Given prosody is available at word level, this information can be incorporated into ASR thorough the LM. In the current work we added an extra feature layer (as shown in Figure 6.1) to the RNNLM to incorporate prosody information into the LM and then later in ASR through decoding (lattice or N–best list rescoring). An n–gram model defines a distribution over discrete symbol sequences which is not the most natural representation for continuously valued prosodic features. Neural network language models [Bengio et al., 2003; Schwenk, 2007; Mikolov et al., 2010, 2011a] transform symbolic word representations to a continuous space, and a number of recent papers have augmented the word feature input to a neural network language model to incorporate additional context [Mikolov and Zweig, 2012] or to learn correlations between words and richer annotations such as part-of-speech tags [Wu et al., 2012]. We have built on these approaches by developing a neural network language model with an extra feature layer to jointly model words and the related prosodic features computed from the context of the current word. We also model prosody at the syllabic level using an automatic syllable detection algorithm discussed in Section 6.2.2 and vary the amount of syllabic context (from 1–10 syllables). We have performed language modelling experiments on the Switchboard and TED corpora reporting results in terms of both perplexity and word error rate (WER) on standard test sets.

### 6.2.1 Previous Work

Prosodic information has been successfully used for topic segmentation [Tür et al., 2001]. In topic segmentation the task is to divide the text into topically homogeneous blocks. Since prosody does not relate to a word identity but directly relates to the discourse structure of speech, it can be expected to signal topic transitions. Through experimental results in [Tür et al., 2001] it was concluded that the prosody features were complementary to the lexical features. Prosody also been used for automatic punctuation by detecting the disfluencies and sentence boundaries [Shriberg et al., 2001].
addition to the punctuation detection, in the same work, prosody was also used to detect the turn takings in multi party meetings [Shriberg et al., 2001]. In other works, prosody was used for dialogue act classification [Shriberg et al., 1998], sentiment classification [Mairesse et al., 2012] and emotion recognition [Luengo et al., 2005]. There have also been a number of attempts to include prosodic information in language modelling.

Vergyri et al. [2003] and Gadde [2000] investigated the use of modelling word durations, using an explicit Gaussian mixture model, trained on feature vectors constructed by considering the duration of the phones in that word; Vergyri et al. also used \( n \)-grams to model the pause duration between the words. Prosodic structure may be interpreted as correlating with non-word phenomena such as sentence boundaries and speech disfluencies including repetitions, deletions and filled pauses; Stolcke et al. [1999] attempted to incorporate this structure using a hidden event \( n \)-gram language model, with prosody of hidden events modelled using a decision tree classifier trained using speech data annotated for sentence boundaries and disfluencies.

Huang and Renals [2007] argued that prosodic information could be more naturally captured at a syllabic level, and used an acoustically-based system for automatic syllable identification. Four-dimensional prosodic feature vectors were extracted, containing F0, energy, slope of F0 and durational information for the syllable, which were vector quantised using a 16-word codebook. Thus a syllable was represented by a single codeword, and a word by a sequence of syllable codewords. This representation is amenable to \( n \)-gram modelling, using factorisation and a hierarchical prior in this case. Maximum entropy language models have also been used to capture prosodic information, such as learning the dependencies between the syntactic features, such as POS tags, and prosodic features, such as accent and duration [Chan and Toggery, 2006].

### 6.2.2 Prosody Feature Extraction

We propose an RNNLM which learns a hidden recurrent state that combines lexical and prosodic information, enabling the RNNLM to learn dependencies between word context and prosodic context. The prosodic contextual features were computed by aligning the transcripts with the corresponding audio (for both training and test data). We used Kaldi based tools [Povey et al., 2011] to align the transcripts. For example if the utterance is *okay all right hey bye-bye*\(^1\) the corresponding word alignments are shown in Figure 6.2. Kaldi based tools provide the alignments at phone level and we

\(^1\)This utterance is selected from Switchboard training corpus
use the information from the lexicon to get alignments at word level. We explored a number of prosodic features in this work outlined in Sections 6.2.2.1 - 6.2.2.5: word duration, pause duration, final phone duration\(^2\) durations and fundamental frequencies computed at syllable level. All duration features were not normalised for speaker, channel, or recording session.

![Figure 6.2: Alignments of an utterance at word level. These alignments are computed by force aligning the transcripts with the acoustic signal.](image)

### 6.2.2.1 Word duration (RNNLM-worddur)

*RNNLM-worddur* models the duration of the previous word, obtained by forced alignment of the training data. For example if the current word is *bye-bye*, we added duration of the previous word *hey* to RNNLM to enhance the context, as shown in Figure 6.3.

### 6.2.2.2 Pause duration (RNNLM-pause)

*RNNLM-pause* models the duration of the pause preceding the current word, obtained by forced alignment of the training data. During modelling if there is no pause we consider the pause duration as zero. For example if the current word in *bye-bye*, we

\(^2\)To model the effect known as *pre-pausal lengthening*
enhanced the context with duration of the pause between the current word (bye-bye) and previous word (hey), as shown in Figure 6.4.

6.2.2.3 Final phone duration (*RNNLM-fphonedur*)

The pause duration between the words affects the duration of the previous word. This effect is known as pre-pausal lengthening [Gadde, 2000]. To model this effect the RNNLMs are given the duration of the final phone in preceding word, as shown in Figure 6.5.

6.2.2.4 Syllable duration (*RNNLM-syldur*)

Huang and Renals [2007] mentioned that modelling the prosody at syllable level is more natural since syllable level prosody features reflect the perceptions of accent, stress and prominence. Aylett [2006] successfully used syllable level prosodic features for dialogue act and hotspot categorisation. The syllable segments were identified using an automatic syllable level detection algorithm described in [Aylett, 2006]. We
Figure 6.4: Pause durations of an utterance computed by force aligning the transcripts with acoustic signal

also used the same setup to detect the syllable-like units\(^3\) to compute the prosody features. The automatic syllable detection algorithm used to detect the syllable-like units consist of two steps. First, a bandpass filter in the range 300–900Hz was used to filter out the energy not belonging to vowels. Second a reverse convex algorithm was used to identify the nuclei of syllables and the corresponding boundaries [Mermelstein, 1975].

For context enhancement experiments we extracted duration of the syllables from the context of the current word, encoded as the index of the previous word and the durations of the syllables in the context. A simple alignment procedure was used to obtain the syllabic context, in which a fixed number of syllable durations preceding the current word are given as an input to the RNNLM, irrespective of sentence boundaries. To investigate the effect of context length, RNNLMs were trained on the durations of the three, five and ten preceding syllables.

\(^3\)We did not use language dependent features to detect the syllables. The main goal was to compute the prosody features for a syllable, not exact boundaries.
6.2. Context-enhancement Using Prosody Features

6.2.2.5 Syllable F0 (RNNLM-sylF0)

Similar to the syllable duration experiments, RNNLMs are given word features and F0 features computed from the syllables in the context of the current word. Four different features are computed at each syllable: mean, maximum, minimum and range of F0 (logarithm is applied to raw F0 reduce the dynamic range). Before computing the features the F0 values of each syllable are normalised using z-score normalisation. To investigate the effect of length of the context the RNNLMs were trained on F0 features computed from three, five and ten preceding syllables.

6.2.3 Text Experiments

Prior to incorporating the prosody RNNLMs into ASR we carried out text experiments to investigate the prediction accuracy of prosody RNNLMs. Our perplexity (PPL) experiments used the Switchboard training transcripts, a total of 3.4M tokens, of which the first 130K tokens were used as validation data, to tune the parameters. We report PPL results using the validation data and the Switchboard evaluation set eval2000.
Chapter 6. Context Enhancement of Recurrent Neural Network Language Models

The eval2000 data contains 20 Switchboard and 20 CallHome English (CHE) conversations. There are approximately 22K tokens in the Switchboard part of eval2000. In this work we report PPLs and WERs on Switchboard conversations only. Hereafter, the Switchboard part of eval2000 is referred as eval2000-swbd.

We estimated a back-off 3-gram LM by interpolating the LMs trained on 3.2M tokens of Switchboard training transcripts and 11M tokens of Fisher English part1 transcripts (LDC2004T19). The LMs are trained using Kneser-Ney smoothing and the interpolation coefficients are optimised for better PPL on validation data. This pruned 3-gram LM is also used in first pass decoding to generate the lattices.

The RNNLMs were trained only on the 3.2M tokens of Switchboard training transcripts. Further details about the acoustic models used to align the data are given in Section 6.2.4.1. A vocabulary of 30,000 words was used, the most frequent words from 3.2M tokens of Switchboard transcripts. The RNNLM used 300 hidden units in the recurrent hidden layer. A factored output layer with 100 classes was used to reduce the computational complexity [Mikolov et al., 2011a]. We modified the Mikolov RNNLM toolkit [Mikolov et al., 2011b] to train the prosody RNNLMs.

Perplexity results for the Switchboard validation and evaluation sets are shown in Figure 6.6. The validation PPLs are in Figure 6.6a and eval2000 PPLs are in Figure 6.6b. These results indicate that the RNNLM improves the perplexity over the 3-gram LM by about 5% relative (on both datasets). The results using the pause duration features result in an improvement of about 13% relative over the baseline RNNLM. The word duration features also reduce the perplexity over the baseline RNNLM, but by a much smaller amount, and the final phone duration features result in an 8–10% relative reduction over the RNNLM baseline. The syllable duration and F0 features result in a reduction of perplexity of over 15% and 13% relative respectively. Similar improvements can be observed after linear interpolation with the 3-gram baseline. The interpolation coefficient was 0.5 in all cases.

6.2.4 ASR Experiments

Following the experiments to investigate the prediction accuracy of prosody RNNLMs, we carried out a set of speech recognition experiments on two tasks: the recognition of Switchboard telephone conversations, and the recognition of TED talks.

The Switchboard telephone conversations were recorded at Texas Instruments dur-
6.2. Context-enhancement Using Prosody Features

Figure 6.6: (a) Perplexities on validation data, (b) Perplexities on Switchboard conversations of eval2000. The Prosody RNNLMs trained on word duration, pause duration, duration of final phone, syllable durations and syllable F0 features. Here the RNNLM-syldur and RNNLM-sylF0 models are trained on syllable context length of five. 3-gram LM is trained on combination of Switchboard and Fisher transcripts and the RNNLMs are trained only on part of Switchboard training transcripts (3.2M).
Chapter 6. Context Enhancement of Recurrent Neural Network Language Models

For ASR experiments we used Switchboard-1 Release 2 (LDC97S62). This release consists of a total of 2400 two-sided telephone conversations, speakers cover all areas of the United States. In the recorded conversations people discussed a total of 70 topics\(^5\), out of which 50 topics were used most frequently. The recordings amount to a total of 300 hours of conversational speech. This release also provided 3.4M tokens of aligned transcripts. In this work we used Mississippi State transcripts\(^6\).

Another task we choose to evaluate the prosody-RNNLMs is the TED task. The description about the TED task and available resources to train acoustic and language models is given in Section 3.2.2. The reason we choose two different tasks to evaluate the prosody RNNLMs is that, different prosodic effects are observed, since Switchboard consists of conversational telephone speech, and TED consists of well-prepared public talks. The language models were evaluated in terms of WER and for each corpus we used standard evaluation protocols: NIST CTS evaluation for Switchboard and IWSLT for TED. The RNNLMs and prosody RNNLMs are incorporated into ASR by rescoring the \(N\)-best list hypotheses, generated from lattices. \(N\)-best list experimental setup is described in Section 3.3.

### 6.2.4.1 Switchboard

We report the WERs on the eval2000-swbd test set, which comprises of a total of 20 conversations, containing 22K word tokens (including sentence start and end symbols) with an out-of-vocabulary (OOV) rate of 5% with respect to our vocabulary of 30K tokens. Two speech recognition acoustic models were used to compute the prosody features: GMM-based and DNN-based. The prosody features were computed by force aligning the transcripts (100-best hypotheses) with the acoustics. We used Kaldi based tools\(^7\) to get the phone, word and pause durations. The Kaldi tools provide the phone durations and we use information from lexicon to get the word durations. We choose two different acoustic models (GMM and DNN) to get the better alignments (the WER of DNN system is lower than the GMM system) and later to compute the precise durational features.

The GMM-based acoustic models were trained on 300 hours of switchboard data, with the transcripts from ISIP of Mississippi State university. The acoustic features comprised 7 frames (±3) of 13-dimensional MFCC features, with the dimension reduced to 40 using a linear discriminant analysis (LDA) transform, followed by a decor-
6.2. Context-enhancement Using Prosody Features

relating semi-tied covariance (STC) transform. The features were adapted per speaker using feature space (constrained) maximum likelihood linear regression (fMLLR). The maximum likelihood system trained on LDA+STC+fMLLR features is then discriminatively trained using the boosted maximum mutual information (bMMI) criteria, with a $b$ value of 0.1. All the baseline experimental results reported here are reproduced using Kaldi speech recognition recipe given in [Veselý et al., 2013].

The DNN-based acoustic models were also trained on 300 hours of Switchboard telephone conversation data [Lu and Renals, 2015]. The acoustic features comprised 11 frames ($\pm 5$) of MFCC features, including delta and acceleration coefficients. The features were transformed using LDA and decorrelated using STC, as for the GMM-based system. However, they were not speaker adapted. The output layer consists of 8827 nodes, corresponding to the set of context-dependent HMM states, used in the GMM-based system. Six hidden layers of 2048 units with a sigmoid non-linearity were used.

The RNNLMs trained on word and prosody features were incorporated into the ASR process by rescoring the 100-best lists, generated from the lattices of the GMM-based system. The prosodic features were computed by aligning the 100-best lists with the acoustics using both GMM-based and DNN-based acoustic models. To compute the final score, the scores of RNNLM are interpolated with the scores of $n$–grams from the 100-best lists. The interpolation coefficients are optimised for better WERs. The WERs computed on Switchboard conversations of eval2000 are given in Table 6.1.

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>GMM(%WER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram</td>
<td>81.9</td>
<td>19.5</td>
</tr>
<tr>
<td>RNNLM</td>
<td>77.5</td>
<td>18.1</td>
</tr>
<tr>
<td>RNNLM-pause</td>
<td>66.5</td>
<td>18.0</td>
</tr>
<tr>
<td>RNNLM-worddur</td>
<td>76.7</td>
<td>17.6</td>
</tr>
<tr>
<td>RNNLM-fphonedur</td>
<td>70.7</td>
<td>17.8</td>
</tr>
</tbody>
</table>

Table 6.1: %WERs computed on 100-best lists of eval2000 data set (Switchboard conversations only). GMM acoustic model is used to force align the transcripts.

In Table 6.1, we can observe that the RNNLM improves the baseline system by 1.4% absolute. In the case of proposed prosody RNNLMs, we can observe 0.5%

\footnote{Due to changes in the Kaldi recipe the reported results here are different to those originally published in [Veselý et al., 2013].}
absolute (3% relative) reduction in WER with RNNLM-worddur and 0.3% absolute with RNNLM-fphonedur models. The improvement with the RNNLM-pause model is much smaller compared to the other models.

In Table 6.2 we report the WERs on the same eval2000 data set but aligned using the DNN-based acoustic models. After aligning the 100-best lists with the DNN-based acoustic models the RNNLM-pause model improves the baseline RNNLM by 0.2% absolute. The accuracy of RNNLM-worddur and RNNLM-fphonedur models were reduced compared to the WERs reported in the second column of Table 6.1.

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>DNN(%WER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram</td>
<td>81.9</td>
<td>19.5</td>
</tr>
<tr>
<td>RNNLM</td>
<td>77.5</td>
<td>18.1</td>
</tr>
<tr>
<td>RNNLM-pause</td>
<td>66.5</td>
<td>17.9</td>
</tr>
<tr>
<td>RNNLM-worddur</td>
<td>76.7</td>
<td>17.9</td>
</tr>
<tr>
<td>RNNLM-fphonedur</td>
<td>70.7</td>
<td>18.0</td>
</tr>
</tbody>
</table>

Table 6.2: %WERs computed on 100-best lists of eval2000 data set (Switchboard conversations only). GMM- and DNN-based acoustic models are used to force align the transcripts.

6.2.4.2 TED talks

Models based on DNNs were used for training the acoustic models for TED [Swietojanski et al., 2013; Bell et al., 2014]. The DNNs were trained using 143 hours of TED lectures, recorded before 2010, and 78 hours of AMI meeting data[^8]. The DNN acoustic features comprised 11 frames (±5) of MFCC features, including delta and acceleration features. The features were transformed using LDA and adapted per speaker using fMLLR. The DNNs were trained by optimising the cross-entropy objective function. There were six 2048-unit hidden layers with sigmoid non-linearity. A pruned 3-gram language model was trained on 312M word tokens[^9] and 2.4M tokens of TED in-domain data was used in the first pass decoding to generate the lattices [Bell et al., 2013a]. The vocabulary consists of 62K tokens (approximately).

[^8]: http://corpus.amiproject.org
[^9]: These tokens were selected from various out-of-domain data sources (Europal (v7), News Commentary (v7), News Crawl (2007 to 2011) and the fifth version of Gigaword) using data selection approach, described in Section 3.2.2
We report PPLs (without interpolation with a 3-gram baseline) and WERs (after interpolation with a 3-gram baseline) on \textit{tst2011} (eight different TED lectures), an evaluation set for IWSLT, consisting of a total of 12K tokens (818 utterances). The RNNLM and prosodically-enhanced RNNLMs were trained using 2.8M tokens, a combination of TED lectures and AMI data (transcripts of 78 hours of AMI data). The development data for tuning the parameters and for early stopping was \textit{tst2012}, also an evaluation set for IWSLT, consisting of 20K tokens. 400 hidden units were again used with a sigmoid non-linearity and a factored output layer with 200 classes was used to reduce the computational requirements.

The WERs of \textit{tst2011} dataset are given in Table 6.3. The RNNLM improves the baseline system by 0.7% absolute (5% relative). As expected, the proposed RNNLM-pause and RNNLM-worddur models are not effective enough to improve the WERs. However, the RNNLM-fphonedur model improves the WERs by 0.2% absolute (2% relative). The small improvements are due to the fact that the prosodic effects/realisation is different in TED talks.

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>%WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram</td>
<td>120.2</td>
<td>12.6</td>
</tr>
<tr>
<td>RNNLM</td>
<td>198.0</td>
<td>11.9</td>
</tr>
<tr>
<td>RNNLM-pause</td>
<td>184.1</td>
<td>12.0</td>
</tr>
<tr>
<td>RNNLM-worddur</td>
<td>194.1</td>
<td>11.8</td>
</tr>
<tr>
<td>RNNLM-fphonedur</td>
<td>184.2</td>
<td>11.7</td>
</tr>
</tbody>
</table>

Table 6.3: PPLs and %WERs computed on \textit{tst2011} and 100-best lists of \textit{tst2011}, respectively. DNN-HMM hybrid acoustic models trained on TED and AMI data are used to force align the transcripts and compute the prosody features.

### 6.2.4.3 Syllable duration and F0

Finally we explored the use of syllable duration and F0 features for the Switchboard task. To investigate the effect of length of syllable context the RNNLM-syldur models were trained on context lengths of 3, 5 and 10. Prior to the computation of syllable level features, the syllable like units were identified using an automatic syllable detection algorithm, described in Section 6.2.2.4. The syllables preceding the current word were identified by using both the syllable and word boundaries (computed by force aligning). 300 recurrent hidden units were used, and a factored output layer with 100
classes was used to reduce the time complexity. From Table 6.4 we can observe 0.3% absolute (2% relative) reduction in WER using the RNNLM trained on durations of 5 syllables (RNNLM-syldur5) from the context of the current word. Similarly we can observe 0.2% absolute improvements with a context length of 3 and 0.1% absolute improvement with a context length of 10.

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>%WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram</td>
<td>81.9</td>
<td>19.5</td>
</tr>
<tr>
<td>RNNLM</td>
<td>77.5</td>
<td>18.1</td>
</tr>
<tr>
<td>RNNLM-syldur3</td>
<td>63.5</td>
<td>17.9</td>
</tr>
<tr>
<td>RNNLM-syldur5</td>
<td>65.0</td>
<td>17.8</td>
</tr>
<tr>
<td>RNNLM-syldur10</td>
<td>63.2</td>
<td>18.0</td>
</tr>
</tbody>
</table>

Table 6.4: %WERs are computed on 100-best lists of eval2000 data set (Switchboard conversations only). Automatic syllable detection algorithm is used to get the boundaries of syllables. GMM-based acoustic models are used to get the word boundary information

Similar to the RNNLM-syldur models the effect of syllable context is investigated by training the RNNLM-sylF0 models on context lengths of 3, 5 and 10. The feature vector is obtained by concatenating the F0 features computed at each syllable in the context\(^{10}\). The features computed are mean, maximum, minimum and range of F0. The hidden layer has 300 hidden neurons and a factored layer with 100 classes was used to reduce the computational complexity. From Table 6.5 we can observe improvements in PPLs but not in WERs.

### 6.3 Context-enhancement Using Syntactic Features

As discussed in Section 6.1 we enhanced the context of a RNNLM by adding a feature layer connecting the hidden layer. In addition to prosody features we also enhanced the context with syntactic features computed from the context of the word. The motivations for using the syntactic features are: First, these features encode some long-distance dependencies, which helps to address the data sparsity issue. Second, in a LM, a word is predicted given the surface form of previous word(s), but there is a huge

\(^{10}\)Kaldi pitch tool is used to compute the F0 features [Ghahremani et al., 2014]
6.3. Context-enhancement Using Syntactic Features

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>%WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram</td>
<td>81.9</td>
<td>19.5</td>
</tr>
<tr>
<td>RNNLM</td>
<td>77.5</td>
<td>18.1</td>
</tr>
<tr>
<td>RNNLM-sylF0_3</td>
<td>66.1</td>
<td>18.4</td>
</tr>
<tr>
<td>RNNLM-sylF0_5</td>
<td>67.3</td>
<td>18.3</td>
</tr>
<tr>
<td>RNNLM-sylF0_10</td>
<td>73.3</td>
<td>18.7</td>
</tr>
</tbody>
</table>

Table 6.5: %WERs are computed on 100-best lists of eval2000 data set (Switchboard conversations only). Before computing the F0 features the F0 sequence of each syllable is normalised using z-score normalisation. GMM-based acoustic models are used to get the word boundary information.

number of possible syntactic features like, part-of-speech tags, lemma, stem, Combinatory Categorial Grammar (CCG) lexical categories (CCG supertags) [Steedman, 2000], features from partial parses of words in the context. Enhancing the context with these features improves the prediction accuracy of the current word being predicted. Third, syntactic features also improve the generalisation to unseen sequences of words during testing, due to the fact that features across different contexts share the information during training. Given the significance of context enhancement, in this work we enhanced the context with part-of-speech (POS) tags and CCG supertags. CCG supertags encode long-distance dependencies which are hard to capture using n-grams. Figure 6.7 presents CCG derivation\[11\] for *I know the boy who Adam thinks Eva believes ate the Apple*. In this example the word *ate* directly depends on the word *boy*. This dependency is captured using the CCG supertag of the word *ate*: $S\backslash NP/\backslash NP$.

In this chapter we have performed experiments to answer two different questions. First, will enhancing the context with syntactic features improve the prediction accuracy and WER of speech recognition? Second, will adapting the tagger (matching the tagger training domain with LM training domain) data improve the speech recognition WERs? At the end we also discuss the experimental results of both POS and CCG supertag.

\[11\]Thanks to Siva Reddy and Bharat Ram Ambati for suggesting me this example and CCG derivation.
6.3.1 Previous Work

In the literature, syntactic features have been successfully used for language modelling. In structured language models (SLMs) [Chelba and Jelinek, 2000] features from a syntactic parse tree were used to improve the language model accuracy. Later [Emami et al., 2003; Emami and Jelinek, 2004] the same features were incorporated into language models based on distributed representations (neural networks). Experimental results revealed that syntactic features significantly reduced the WERs of WSJ and Switchboard speech recognition tasks. Mirowski and Vlachos [2015] proposed dependency recurrent neural network language model (Dependency RNN). In this work the RNN was trained on different paths that exist in a dependency parse tree. The proposed Dependency RNNs were used for the Microsoft sentence completion challenge and showed significant improvements in terms of accuracy. Similar to the dependency RNNLMs, dependency tree paths were explored in the context of n-grams also [Gubbins and Vlachos, 2013]. Wu et al. [2012] have shown that Part-of-Speech (POS) tags used as a factor to RNNs improve the performance of language modelling. On the IWSLT TED test data sets, they obtained significant reductions in the perplexity and WERs. In top-down tree LSTMs the probability of a sentence is equal to the probability of generating the corresponding dependency tree [Zhang et al., 2016]. These models were also employed in the Microsoft sentence completion challenge and have shown significant improvements. Syntactic language models have also been used in other tasks like machine translation [Sennrich, 2015]. Given the significance of syn-
tactic language models, in this work we used the syntactic information in the context of language models for automatic speech recognition.

### 6.3. Part-of-Speech (POS) Tags

POS tagging is the task of assigning a part-of-speech (POS) tag for each word in a sentence. The words in natural language are ambiguous and POS tags can help in disambiguating them. For example, ‘book’ can be a noun as in ‘I read a book’ or a verb as in ‘Book a flight’. Also, only particular sequences of POS tags are possible in a language which can also be useful for language modelling. For example a determiner is followed by an adjective or noun but not by a verb. This information can be useful in handling the sparsity in the data.

Wu et al. [2012] have shown that POS tags used as a factor to RNNs improve the performance of language modelling. They provided POS tags from Genia tagger [Tsuruoka05, TsuruokaT05] as a factor to RNNs. In this work, we extend work of Wu et al. [2012] both in terms of speed and accuracy. It took 18 mins to tag the 7.5M tokens of TED training data using Genia tagger, which is very slow. Instead of Genia tagger, we use C&C POS tagger [Clark and Curran, 2007] in our experiments. C&C tagger took 20 seconds to tag the same 7.5M tokens of TED training data which is 50 times faster than the Genia tagger. The C&C tagger has competitive performance compared to Genia tagger and since it is faster, it is a more feasible option to use in real-world speech recognition applications.

Most of the work in statistical POS tagging uses WSJ sections of Penn Treebank [Paul and Baker, 1992], which is newspaper text. A significant drop in performance is observed when a tagger trained on WSJ corpus is tested on data from a different domain like the Brown corpus, Bio-medical, Twitter data sets [Curran and Clark, 2003; Tsuruoka et al., 2005; Tsuruoka and Tsujii, 2005; Gimpel et al., 2011]. There has been lot of research on adapting taggers to different domains [Tsuruoka and Tsujii, 2005; Gimpel et al., 2011]. A widely used technique for domain adaptation is adding domain specific data to WSJ corpus while training the tagger. For example, when the Genia tagger trained on WSJ corpus is tested on the GENIA corpus, accuracy of the tagger is 85.19%. Adding the bio-medical corpus (the GENIA corpus and the PennBioIE corpus) to WSJ corpus while training boosted the tagger performance to 98.26%. Though the Genia tagger improved the performance of RNN based language models, the tagger is trained on WSJ and bio-medical corpus which is different from
the testing data of TED talks. In this work we adapt the C&C tagger to TED talks by adding different data sets to WSJ corpus while training. We explore the usefulness of two other sections of the Penn Treebank which are the Brown corpus [Francis and Kucera 1979] and the Switchboard corpus [Godfrey et al. 1992]. Brown corpus is a one million word corpus of American English from the 1960s. Unlike WSJ corpus, Brown corpus contains text from different genres like press, religion, fiction. We also explore a self-training technique for domain adaptation. In this approach, we first run the C&C tagger trained on WSJ corpus to tag the 7.5M token TED talks training data. We then add the tagged TED talks training data to WSJ corpus and re-train the C&C tagger. In addition to differences in domains, we also observed slight variations (for example, capitalisation) in the tokenisation between WSJ corpus and TED talks data. To handle these variations, we tokenised TED talks data using Penn Treebank style tokeniser. We also normalised Penn Treebank data to all lower case since TED talks data is in all lower case.

6.3.3 CCG Supertags

In addition to POS tags, we also experimented with CCG supertags derived from C&C supertagger [Clark and Curran 2007]. CCG is an efficiently parseable, yet linguistically expressive grammar formalism. It has a transparent interface between surface syntax and underlying semantic representation, including predicate-argument structure, quantification and information structure. In CCG, words are associated with syntactic categories (supertags), which are either simple such as NP, S or complex such as (S\NP)/NP. Complex categories of the form X\Y or X/Y are functors, which respectively take an argument Y to their left or right (depending on the direction of the slash) and yield a result X.

The C&C supertagger assigns a CCG supertag to a word based on the context. Unlike a POS tag, a supertag contains rich sub-categorisation information. For example, the supertag for the transitive verb is (S\NP)/NP, which looks for an object NP to the right and then a subject NP to the left to form a sentence S. Similarly, supertag for an adjective is (N/N), which demands a noun to its right. In this way, supertag helps in identifying the expectation on the next word and we hope such information can help in language modelling. We used first best supertag from the C&C supertagger [Clark and Curran 2007] whose accuracy on WSJ development set is 91.5%. The POS and CCG supertags for a sentence are shown in Table 6.6.
6.3.4 ASR Experiments

The proposed syntactic RNNLMs are incorporated into ASR by rescoring the $N$-best lists ($N=100$) of each utterance. The experimental setup to rescore the $N$-best lists is described in Section 3.3.

6.3.4.1 TED ASR task

The task we chose to evaluate the syntactic RNNLMs is TED task. The description about the TED task and available resources to train acoustic and language models are given in Section 3.2.2. In this work, we used the dev2010 and tst2010 sets for development and the tst2011, tst2012 and tst2013 sets for evaluation. To obtain maximum gains we always interpolate the scores of $n$-grams and RNNLMs. We used combination of dev2010 and tst2010 for early stopping and to tune the interpolation coefficients.

6.3.4.2 Acoustic Models

ASR system uses deep neural networks (DNNs) for training the acoustic models. The DNNs are trained in hybrid configuration. The hybrid configuration use the Multi Level Adaptive Neural Network (MLAN) architecture for domain adaptation, described in Section 4.5.2. Hybrid acoustic models were trained on 143 hours of TED data and 127 hours of AMI data. The hybrid DNN consists of six hidden layers, each layer consisting of 2048 hidden neurons.

6.3.4.3 $N$–grams

The ASR system uses Kneser-Ney (KN) smoothed $n$–gram language models for lattice rescoring and decoding. Using the language resources described in Section 3.2.2.2 we trained the $n$–grams on 2.4M tokens of in-domain and 312M tokens of OOD data. The final $n$–gram language model is obtained by interpolating in-domain and OOD LMs. The interpolation coefficients were optimised on development data (combination of
dev2010 and tst2010). In the final ASR system KN smoothed 3-gram and 4-gram LMs are used for decoding and lattice rescoring, respectively.

The vocabulary of this task is 62,522, which consists of all the words in in-domain data except the words with a frequency of one and all the words exceeding specified occurrence count thresholds in the OOD data.

### 6.3.4.4 RNNLMs

RNNLMs were also trained on a combination of in-domain and OOD data. As mentioned above, a total of 312M tokens were selected from OOD data. Given complexity of training RNNLMs, we have trained the RNNLMs and syntactic RNNLMs on a combination of in-domain and different smaller subsets of OOD data. Again we used the cross-entropy difference (CED) metric to create subsets of OOD data. A total of two OOD subsets were selected from 312M OOD tokens, by controlling the CED metric threshold. The number of tokens after combining with the OOD data and the corresponding vocabulary sizes are given in Table 6.7. The vocabulary to train the RNNLM consists of all the distinct words from in-domain and most frequent words in the OOD data. Remaining words in OOD data, which were not part of the vocabulary are replaced with special token, <unk>.

<table>
<thead>
<tr>
<th>#Words</th>
<th>#Sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4M</td>
<td>749.3K</td>
</tr>
<tr>
<td>12.4M</td>
<td>1298.8K</td>
</tr>
</tbody>
</table>

Table 6.7: Number of tokens after combining the in-domain data with subsets of OOD data. In-domain data consists of 2.4M tokens.

We used Mikolov RNNLM toolkit [Mikolov et al., 2011b] to train the baseline RNNLMs and with POS features. The RNNLMs are trained on POS features by adding an extra feature layer connecting the hidden layer, as shown in Figure 6.1. 1-of-k coding\footnote{Only value of index representing POS tag of the previous word is one and all other values are zero} is used to convert the POS tags to a feature vector. The vocabulary of the POS tags is 50 and the dimensionality of POS feature vector is also the same. The neural nets are trained until the difference in entropy between two successive iterations is less than a predefined threshold. In all the experiments reported in this work the threshold is 0.01. The validation data for early stopping – the combination of dev2010
and tst2010 – contains a total 44K tokens. We used starting learning rate of 0.1 for both the models.

6.3.4.5 Experimental Results

The WERs of 4-gram and RNNLM baselines trained on 7.4M and 12.4M tokens are given in Table 6.8. From the Table 6.8 we can observe that the improvements with the RNNLMs trained on 7.4M (4-gram+RNNLM-7.4M) tokens are in the range of 0.3% to 0.4% absolute (1.7% to 6% relative). Similarly, with an RNNLM trained on 12.4M tokens the improvements are in the range of 0.5% to 1.0% absolute.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4-gram</td>
<td>15.1</td>
<td>13.5</td>
<td>11.2</td>
<td>12.1</td>
<td>22.4</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M</td>
<td>14.8</td>
<td>13.1</td>
<td>10.5</td>
<td>11.6</td>
<td>22.0</td>
</tr>
<tr>
<td>4-gram+RNNLM-12.4M</td>
<td>14.6</td>
<td>12.9</td>
<td>10.2</td>
<td>11.4</td>
<td>21.9</td>
</tr>
</tbody>
</table>

Table 6.8: The WERs of baseline 4-gram and RNNLMs trained on 7.4M and 12.4M tokens

We did different POS adaptation experiments on the 7.4M word training data. We first provided the POS tags from Genia tagger as a feature to RNNLM. Results of this experiment are presented in the third row of Table 6.9. As mentioned previously, the Genia tagger is very slow and also uses biomedical data for training which might not be very useful for testing TED talks. Next, we experimented with C&C tagger trained on WSJ corpus. The fourth row of Table 6.9 presents these results. This experiment gave slightly better results compared to Genia tagger since C&C tagger is better than Genia tagger. Then we experimented with adding the Brown corpus and the Switchboard corpus to the WSJ corpus for training the POS tagger. We obtained the best results with C&C tagger trained on both WSJ and Switchboard corpus. Switchboard corpus gave slightly better results compared to Brown corpus, shown in the fifth and sixth rows of Table 6.9. This shows the importance of domain for training the POS tagger. Training the POS tagger on POS tagged data of TED talks would be ideal as we will be training and testing on the same domain. Since we don’t have POS tags for TED talks, we did a self-training experiment where we tagged the TED talks data with C&C tagger trained on WSJ and used this data to re-train the tagger. Since the POS tags of TED training data in this self-training approach is not accurate we didn’t observe any
improvements over the model trained on Switchboard corpus, as seen in the seventh row of Table 6.9.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4-gram</td>
<td>15.1</td>
<td>13.5</td>
<td>11.2</td>
<td>12.1</td>
<td>22.4</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M</td>
<td>14.8</td>
<td>13.1</td>
<td>10.5</td>
<td>11.6</td>
<td>22.0</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M-Genia</td>
<td>14.8</td>
<td>12.9</td>
<td>10.5</td>
<td>11.5</td>
<td>22.0</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M-C&amp;C (WSJ)</td>
<td>14.7</td>
<td>12.8</td>
<td>10.4</td>
<td>11.6</td>
<td>22.0</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M-C&amp;C (WSJ + Brown)</td>
<td>14.7</td>
<td>12.8</td>
<td>10.4</td>
<td>11.5</td>
<td>22.0</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M-C&amp;C (WSJ + Switchboard)</td>
<td><strong>14.7</strong></td>
<td><strong>12.7</strong></td>
<td><strong>10.3</strong></td>
<td><strong>11.4</strong></td>
<td><strong>22.0</strong></td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M-C&amp;C (WSJ + TED)</td>
<td>14.7</td>
<td>12.9</td>
<td>10.5</td>
<td>11.4</td>
<td>22.0</td>
</tr>
</tbody>
</table>

Table 6.9: The WERs of RNNLM trained on POS tags. Here the taggers are adapted to the language model training data.

In this section first we did experiments with Genia and C&C taggers on 7.4M words training data and obtained best results with C&C tagger trained on WSJ and Switchboard corpus. To observe the impact of the POS tags in the context of more training data, we took the model which gave best results on 7.4M words training data and replicated it on the 12.4M training data. Table 6.10 presents the results of adding POS tags from C&C tagger trained on WSJ and Switchboard using 7.4M and 12.4M training data. We observed consistent improvements across all the test sets and training data. Improvements by adding POS tags for larger training data (12.4M) is slightly lower compared to smaller training data (7.4M).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4-gram+RNNLM-7.4M</td>
<td>14.8</td>
<td>13.1</td>
<td>10.5</td>
<td>11.6</td>
<td>22.0</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M (WSJ + Switchboard)</td>
<td><strong>14.7</strong></td>
<td><strong>12.7</strong></td>
<td><strong>10.3</strong></td>
<td><strong>11.4</strong></td>
<td><strong>22.0</strong></td>
</tr>
<tr>
<td>4-gram+RNNLM-12.4M</td>
<td>14.6</td>
<td>12.9</td>
<td>10.2</td>
<td>11.4</td>
<td>21.9</td>
</tr>
<tr>
<td>4-gram+RNNLM-12.4M (WSJ + Switchboard)</td>
<td><strong>14.5</strong></td>
<td><strong>12.7</strong></td>
<td><strong>10.2</strong></td>
<td><strong>11.4</strong></td>
<td><strong>21.8</strong></td>
</tr>
</tbody>
</table>

Table 6.10: WERs of RNNLMs trained on 7.4M and 12.4M tokens and their POS tags.

### 6.3.4.6 CCG Supertags

Similar to POS tag experiments, we also did experiments with CCG supertags. Since we have CCG supertags only for the WSJ corpus, we could do any adaptation experi-
ments. We did three different experiments with CCG supertags. In exp-1, we assigned CCG supertags to all the words. In exp-2 we assigned the CCG supertag if it occurred more than \( k \)–times in the data. For the rest of the words we assigned the POS tag of that word. We experimented with different values of \( k \): 10, 20, 50, 100. Through experimental results we found that \( k=50 \) gave better results.

Table 6.11 presents the results of CCG supertag experiments. Exp-2 where we assigned CCG supertags over a threshold gave slightly better results compared to exp-1. Since there are 340 different supertags keeping a threshold seems useful. But CCG supertags didn’t improve over baseline RNN. Since the CCG supertagger is trained on WSJ corpus and the testing data is TED talks which is out-of-domain data, CCG supertags assigned by the supertagger might not be very accurate which could be the reason for no improvements in WER.

<table>
<thead>
<tr>
<th>Model</th>
<th>dev2010</th>
<th>tst2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-gram</td>
<td>15.1</td>
<td>15.1</td>
</tr>
<tr>
<td>4-gram+RNNLM-7.4M</td>
<td>14.8</td>
<td>14.6</td>
</tr>
<tr>
<td>4-gram+RNNLM-CCG (exp-1)</td>
<td>15.1</td>
<td>15.2</td>
</tr>
<tr>
<td>4-gram+RNNLM-CCG (exp-2)</td>
<td>15.1</td>
<td>15.1</td>
</tr>
</tbody>
</table>

Table 6.11: WERs of \textit{dev2010} and \textit{tst2010} using models trained on CCG Supertags

6.4 Discussion

6.4.1 Prosody Features

From the experimental results given in Table 6.1 we can observe moderate improvements with the RNNLMs trained on prosodic features over the baseline RNNLM. We can observe significant improvements with the RNNLM-worddur models compared to the other models. The RNNLM-worddur model improves the baseline by reducing the number of deletions. In the case of RNNLM-pause model, surprisingly there is no correlation between the PPL and WER improvements. A possible reason for this behaviour is that the \( N \)–best lists can be noisy and it is sometimes difficult to get precise alignments to the pause regions. In Table 6.2, we can observe another 0.1% absolute with the pause features computed by DNN-based hybrid acoustic models.
From Table 6.4 we can observe that the length of the context has an effect on percentage of errors. As the length of the context increased the accuracy of the models reduced (RNNLM-syldur10). Given RNNLM-sylF0 models (in Table 6.5) are doing worse than baseline models further experiments can be done by training the models on a combination of syllable duration and F0 features. Here the features are normalised by z-score normalisation; more experiments can be done by training the models on unnormalised features or normalised by other techniques. From Table 6.3 we can observe that the proposed models fail to improve upon the baseline model possibly since the task is recognising the lectures and prosodic variations are fewer compared to the Switchboard conversations. In this work, the effect of pause duration on the preceding word is modelled by training the RNNLMs on the duration of the final phone in the preceding word. This effect can be further investigated by training the RNNLMs on the duration of the final vowel or syllable in the preceding word.

To find the statistical significance of improvements, we performed matched pair sentence segment word error (MPSSWE) [Gillick and Cox, 1989] tests for the proposed prosody RNNLMs and the baseline, given in Table 6.1. The statistical significance test reveals that the improvements with RNNLM-worddur and RNNLM-fphonedur models are statistically significant at $p < 0.01$ and $p < 0.05$, respectively. Where as the improvement with RNNLM-pause is not statistically significant.

At test time the prosody features were computed by aligning the first-pass/100-best hypotheses with the corresponding audio. The errors in the first-pass/100-best hypotheses might effect the computation of prosody features. To investigate the effect of errors on computation of prosody features we compared the alignments of true transcripts with the alignments obtained with the ASR hypotheses. The observation is that all OOV words in the hypotheses were converted to &lt;unk&gt; and the &lt;unk&gt; symbol is mapped to the silence phone. So all the OOV words were treated as silence. I think this is one of the effects on computation of prosody features and this is evident if we compare the PPL improvement of RNNLM-sildur model in Figure 6.6 with WER of RNNLM-sildur model in Table 6.1. Since an error word is mapped to a silence phone this is also effects the computation of word and final phone durations.

---

13This is a default option in Kaldi tools
6.4.2 Syntactic Features

From Table 6.9 we can observe that after adaptation the improvements are moderate (0.1% to 0.4% absolute) and consistent across datasets, except tst2013. We looked at the alignments of each hypothesis with the reference to find where POS information helping to reduce the errors. An example utterance is given in Table 6.12. The correct word in the reference is ‘magnets’ but RNNLM generated ‘magnet’ which is wrong. Adding POS tags to RNNLM generated the correct word ‘magnets’. POS tags for ‘nine’ and ‘thousand’ are CD. POS tag of ‘magnets’ is NNS (plural form) whereas the POS tag of ‘magnet’ is NN (singular form). A sequence of CD tags are followed by a plural form noun NNS rather than a single form noun NN. With the help of POS tags the model could learn this information and was able to generate the correct word.

Reference: in one of the joints between over nine thousand magnets in lhc there was a manufacturing defect

RNN: in one of the joins between over nine thousand magnet in lhc there was a manufacturing defect

RNN + POS: in one of the joins between over nine thousand magnets in lhc there was a manufacturing defect

Table 6.12: The utterances where POS tags helped to select a better hypothesis than the baseline RNNLM

From the experimental results given in Table 6.10, we can observe that the % improvements with models trained on 12.4M tokens and their POS tags is less than the with models trained on 7.4M tokens. The reason could be that some of the information provided by POS tags is being captured by adding more training data.

As discussed in Section 6.3, the CCG supertags encode the long distance dependencies and we expected good gains with these features. But from the experimental results given in Table 6.11 we can conclude that these features are not effective enough to improve the RNNLM baselines. One reason could be that RNNLMs over-fit the CCG Supertags.\footnote{PPL of RNNLMs trained on Supertags is far lower than the RNNLM baseline}
6.5 Summary

In this chapter we enhanced the context with prosody and syntactic features computed from the context of the current word. To investigate enhancement using prosodic features, we did experiments using previous word duration, pause duration between the current and previous word, final phone duration preceding the pause duration, syllable duration and syllable F0. From the experimental results we found that the duration features (both word syllable) out-perform the other features, in terms of PPL and WERs. As expected the gains on Switchboard task are higher than the TED task. To investigate the use of syntactic features, we enhanced the context with POS and CCG Supertags. We also did experiments by adapting the POS tagger. From the experimental results the conclusions are improvements with before and after POS tagger adaptation are moderate and the CCG supertags are doing worse than the baseline.
Chapter 7

Conclusion and Future Work

In this chapter, we summarise all the work and also describe the main conclusions of this thesis. At the end we outline some future directions of this work.

7.1 Summary and Conclusion

This thesis is concerned with exploring recurrent neural network language models (RNNLMs) for large vocabulary continuous speech recognition (LVCSR). We have conducted experiments to investigate (1) better learning procedures for RNNLMs (pre-training algorithm); (2) whether enhancing the context of RNNLM improves the prediction accuracy (with prosody and syntactic features); (3) adaptation methods to adapt the RNNLMs to a target domain at test time.

We proposed a novel pre-training learning procedure for RNNLMs by sharing the output parameters with the NNLM, in Chapter 4. In Chapter 5 we described our investigation into adaptation of RNNLMs to a target domain using two approaches: 1) by learning hidden unit contributions (LHUC) 2) by adapting all the parameters. Finally, in Chapter 6 we have reported experiments to investigate the context enhancement of RNNLMs using prosody and syntactic features.

The major conclusions and observations of this thesis are:

- Our experiments reported in Chapter 4 investigated whether transfer of information from a trained feed-forward neural network language model (NNLM) can be used to initialise the parameters of the RNNLM. From the experimental results given in Figure 4.8 and Figure 4.9 we can conclude that the proposed pre-training has been useful for ASR, in terms of WER. One for this reason
could be that initialising the output layer parameters of the RNNLM with the
NNLM forces the RNNLM to initially to have same hidden representations as
that of the feed-forward NNLM (this is evident from Figure 4.7, drop in PPL in
initial iterations of pre-trained LM). Pre-training also helps to converge to a bet-
ter local minima than RNNLMs, which were not pre-trained. Overall absolute
WER improvements after pre-training are in the range of 0.1% to 0.4%. We have
also conducted PPL based experiments to investigate convergence of pre-trained
models and the minimum number of pre-training iterations required to improve
the prediction accuracy. From the experimental results given in Figure 4.7 we
can conclude that pre-trained RNNLMs converge faster than the RNNLMs. Also
from Figure 4.5 and Figure 4.6 we can conclude that a few iterations of pre-
training is sufficient to improve the prediction accuracy.

• Since the language model training data may not cover all the domains in the test
data, sometimes we need to adapt the LMs. We investigated two approaches to
adapt the RNNLMs: (1) learning hidden unit contributions (RNNLM-LHUC)
(2) adapt all the parameters (RNNLM-adapt-all). The conclusions from exper-
imental results given in Tables 5.1 and 5.3 are that adaptation improves the WER
but the improvements are small although statistically significant. We also found
that there is no correlation between the % improvements and amount of LM
training data (Tables 5.2 and 5.4). We carried out further experiments by adapting
only a subset of parameters of the RNNLM. We observed that adapting only
recurrent weights improved WER but there is no effect when other weights are
adapted (Table 5.5). Since we are adapting the RNNLMs during test time using
small amounts of adaptation data, we looked at whether the models over-fits the
adaptation data or not. The observations are RNNLM-LHUC does not over-fit
the adaptation data - the adaptation parameters are far less than the parameters of
RNNLM. However, RNNLM-adapt-all over-fits the adaptation data (Table 5.7).
By looking at alignments of reference and hypothesis transcripts we concluded
that, as the number of topics in the show (television programme used for test
data) increases, the adaptation had a negative effect on WER. From the exper-
imental results given in Table 5.8 we can conclude that the improvements with
the supervised adaptation are similar to that of unsupervised adaptation and that
supervised adaptation requires large amounts of human-annotated data.

• Language model prediction accuracy can be improved by providing more con-
textual information. In Chapter 6 we report experiments investigating the effect of providing more context information to the RNNLMs using prosody and syntactic features, computed from the context of the current word. From the PPL experiments (Figure 6.6) we found that all the prosody features improved the prediction accuracy. The pause duration model outperformed (in terms of PPL) the other features with a relative improvement of 13%. The ASR experiments using prosody features revealed that the duration features outperform the other prosody features (Tables 6.1 and 6.4). Given that word duration features outperform the other features, we looked at alignments of reference with the hypothesis and concluded that the word duration model prefers longer utterances by reducing the number of deletions. However ASR improvements are contrary to the PPL improvements, specifically in case of pause duration feature. One reason could be that N–best lists are noisy and computation of exact pause duration is quite difficult. As described in Section 6.2.2 the prosody features were computed by aligning the transcripts with the acoustics. We used both the GMM-HMM and DNN acoustic models for computation of prosody features. From Table 6.2, the observation is that even though DNN acoustic models provide better alignments the improvements are relatively small compared to GMM alignments (Figure 6.1). The reason could be that the fundamental difference in modelling the acoustics by GMM-HMMs and DNNs, and we rescored the N–best lists generated by a GMM-HMM system. We got improvements with syllable duration but not with the F0 features computed at the syllable level (Table 6.5). This might due to the basic unit we chose to compute the F0 and the conclusion is further exploration is required by computing the F0 features at a basic unit longer than the syllable. Finally, the relative improvements of the TED task are less than the Switchboard task (Tables 6.1 and 6.3). This is due to the fact that the Switchboard data is more conversational than the TED lectures.

In addition to prosody features we also enhanced the context with syntactic features, in Section 6.3. We conclude that part-of-speech (POS) tag features are helpful for ASR in terms of WERs, from Table 6.9. The improvements are moderate and consistent across datasets. Observing the alignments of reference with the hypothesis revealed that the POS features are helping to reduce the specific errors (Section 6.4). The improvements with the POS features are vanishing with respect to language model training data, from Table 6.10. Even though CCG supertags encode the long distance dependencies, these features are not effective
enough to improve over the RNNLM baseline (Table 6.11).

To conclude, the proposed advances to the RNNLMs (described in Chapter 1) have shown significant improvements in terms of WERs.

7.2 Future Work

In continuation to the work in this thesis we have interesting ideas worth investigating in future.

• In Chapter 4 we have conducted pre-training experiments by sharing the output layer parameters. It would be worth investigating the concept of co-training. In co-training both the NNLM and RNNLM are jointly trained.

• Given that the prosody RNNLMs were trained only on 3.2M tokens of acoustic data (in Chapter 6), it is worth investigating how well the prosody RNNLMs trained on much more data are complementary to the baseline models, also trained on more data. Here the current word was predicted given the features computed from the previous word. In addition to this experiment, it would be good to investigate the prediction of the current word given the features computed from the words both preceding and following the current word. Finally, the prosody features and words are alternatively modelled using multi-task learning [Caruana 1997; Seltzer and Droppo 2013]. In multi-task learning an extra output layer can be added to predict the prosodic features. During testing the network only predicts the probability distribution over the words.

• In the current unsupervised adaptation approach (in Chapter 5), we gave equal weight to both correctly recognised and misrecognised words in the 1-best decoding. The influence of errors during adaptation could be reduced by scaling the gradients in proportion to confidence scores of each word. Since we observed over-fitting with adapt-all the parameters adaptation method (in Table 5.7), there is some potential in combining this method of adaptation with larger rates and appropriate regularisation (e.g. KL-divergence regularisation [Yu et al. 2013; Liu et al. 2016a]) or confidence measures. Finally, significant amounts of manually generated metadata are available for broadcast transcription (Section 3.2.3) and it should be possible to exploit this information to better aid the adaptation process.
7.2. Future Work

Following immediate extensions to the work in this thesis, we have interesting future directions to further explore the language models for speech recognition.

- We know that the RNNLNs learns the long-distance dependencies. However it is not completely understood how they learn these dependencies. It would be worth investigating attention based approaches to reveal this [Mnih et al. 2014; Johnson et al. 2016].

- The RNNLNs or its variants (LSTMs, GRUs) have shown significant improvements in speech recognition. But these models are incorporated into ASR by rescoring lattices based on some approximations in the second pass decoding [Hori et al. 2014; Liu et al. 2016b]. We could achieve further gains by incorporating these models in the first pass decoding, by addressing huge number of computations.

- Given there are commonalities across languages (for example Hindi and Urdu or Kannada and Telugu [Reddy and Sharoff 2011; Swietojanski et al. 2012]), it would be worth looking into how can we make use of commonalities in one language while training the language models for another language, using neural networks.

- Since these models have a huge number of parameters they can’t be used in devices with less memory. It would be worth investigating using teacher-student based learning approaches for efficient use of these models in the devices with less memory [Hinton et al. 2015; Jinyu Li 2014].
Appendix A

A.1 Show level WERs of dev.full

In Chapter 5, we investigated the unsupervised adaptation of RNNLMs to a specific BBC show and the average WERs on dev.full and eval.task1 are given in Table 5.1 and 5.3. Since the RNNLMs were adapted to a specific BBC show, the WERs before and after adaptation are given in Tables A.1 and A.2. The maximum absolute WER gains at show level with LHUC and Adapt-all approaches are 0.5 and 0.9, respectively.

<table>
<thead>
<tr>
<th>Show name</th>
<th>Baseline</th>
<th>LHUC</th>
<th>Adapt-all</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbcfour_the_book_quiz</td>
<td>24.9</td>
<td>24.8</td>
<td>24.6</td>
</tr>
<tr>
<td>bbctwo_snooker_world_championship</td>
<td>29.9</td>
<td>29.9</td>
<td>29.8</td>
</tr>
<tr>
<td>bbcone_panorama</td>
<td>13.5</td>
<td>13.4</td>
<td>13.4</td>
</tr>
<tr>
<td>bbcthree_glamour_girls</td>
<td>34.4</td>
<td>34.3</td>
<td>34.3</td>
</tr>
<tr>
<td>bbcfour_christina_a_medieval_life</td>
<td>16.4</td>
<td>16.3</td>
<td>16.4</td>
</tr>
<tr>
<td>bbcone_waking_the_dead</td>
<td>33.5</td>
<td>33.3</td>
<td>33.3</td>
</tr>
<tr>
<td>bbcthree_placebo</td>
<td>43.4</td>
<td>43.5</td>
<td>43.5</td>
</tr>
<tr>
<td>bbctwo_schoolslet_s_write_poetry</td>
<td>18.7</td>
<td>18.5</td>
<td>18.3</td>
</tr>
<tr>
<td>bbcthree_scallywagga</td>
<td>44.6</td>
<td>44.3</td>
<td>44.7</td>
</tr>
<tr>
<td>bbctwo_a_picture_of_britain</td>
<td>15.2</td>
<td>15.0</td>
<td>15.2</td>
</tr>
<tr>
<td>bbctwo_escape_to_the_country</td>
<td>14.7</td>
<td>14.7</td>
<td>14.6</td>
</tr>
<tr>
<td>bbcone_meet_the_immigrants</td>
<td>50.2</td>
<td>50.1</td>
<td>50.2</td>
</tr>
</tbody>
</table>

Table A.1: % WERs of dev.full at show level: before and after adaptation using LHUC and Adapt-all approaches
<table>
<thead>
<tr>
<th>Show name</th>
<th>Baseline</th>
<th>LHUC</th>
<th>Adapt-all</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbctwo_schools_primary_geography</td>
<td>42.0</td>
<td>41.6</td>
<td>41.1</td>
</tr>
<tr>
<td>bbctwo_great_british_menu</td>
<td>21.7</td>
<td>21.7</td>
<td>21.7</td>
</tr>
<tr>
<td>bbcone_bill_oddie_s_wild_side</td>
<td>35.9</td>
<td>35.6</td>
<td>35.2</td>
</tr>
<tr>
<td>bbthree_dog_borstal</td>
<td>28.3</td>
<td>28.5</td>
<td>28.5</td>
</tr>
<tr>
<td>bbctwo_something_special</td>
<td>43.4</td>
<td>43.3</td>
<td>43.5</td>
</tr>
<tr>
<td>bbcone_to_buy_or_not_to_buy</td>
<td>23.1</td>
<td>22.9</td>
<td>22.8</td>
</tr>
<tr>
<td>bbcone_doctors</td>
<td>36.1</td>
<td>36.0</td>
<td>36.2</td>
</tr>
<tr>
<td>bbcone_space_pirates</td>
<td>42.4</td>
<td>42.3</td>
<td>42.0</td>
</tr>
<tr>
<td>bbctwo_through_the_keyhole</td>
<td>21.9</td>
<td>21.9</td>
<td>22.1</td>
</tr>
<tr>
<td>bbctwo_ready_steady_cook</td>
<td>32.8</td>
<td>32.8</td>
<td>32.7</td>
</tr>
<tr>
<td>bbcone_newsround</td>
<td>26.6</td>
<td>26.4</td>
<td>26.1</td>
</tr>
<tr>
<td>bbctwo_eggheads</td>
<td>16.3</td>
<td>16.2</td>
<td>16.4</td>
</tr>
<tr>
<td>bbcfour_world_news_today</td>
<td>13.1</td>
<td>13.2</td>
<td>13.2</td>
</tr>
<tr>
<td>bbcone_the_one_show</td>
<td>22.2</td>
<td>22.4</td>
<td>22.4</td>
</tr>
<tr>
<td>bbctwo_the_trees_that_made_britain</td>
<td>26.4</td>
<td>26.3</td>
<td>26.1</td>
</tr>
<tr>
<td>bbcone EASTENDERS</td>
<td>38.2</td>
<td>38.0</td>
<td>38.2</td>
</tr>
<tr>
<td>bbcone_a_question_of_sport</td>
<td>41.9</td>
<td>41.4</td>
<td>41.8</td>
</tr>
<tr>
<td>bbctwo_sportsround</td>
<td>30.9</td>
<td>30.6</td>
<td>30.8</td>
</tr>
<tr>
<td>bbcone_doctor_who</td>
<td>44.9</td>
<td>44.5</td>
<td>44.6</td>
</tr>
<tr>
<td>bbctwo_dad_s_army</td>
<td>45.1</td>
<td>44.7</td>
<td>45.2</td>
</tr>
<tr>
<td>bbcthree_doctor_who_confidential</td>
<td>33.2</td>
<td>33.2</td>
<td>33.0</td>
</tr>
<tr>
<td>bbc_three_top_gear</td>
<td>29.8</td>
<td>29.7</td>
<td>29.7</td>
</tr>
<tr>
<td>bbc_three_two_pints_of_lager_and</td>
<td>34.3</td>
<td>34.0</td>
<td>33.8</td>
</tr>
<tr>
<td>bbctwo_fimbles</td>
<td>43.3</td>
<td>43.1</td>
<td>42.8</td>
</tr>
<tr>
<td>bbctwo_tikkabilla</td>
<td>28.4</td>
<td>28.0</td>
<td>27.9</td>
</tr>
<tr>
<td>bbctwo_premiership_rugby</td>
<td>51.5</td>
<td>51.4</td>
<td>51.3</td>
</tr>
<tr>
<td>bbcone_bbc_london_news</td>
<td>16.6</td>
<td>16.7</td>
<td>16.5</td>
</tr>
<tr>
<td>bbc_three_radio_1_s_big_weekend</td>
<td>51.0</td>
<td>51.0</td>
<td>50.8</td>
</tr>
<tr>
<td>bbc_three_little_britain</td>
<td>41.6</td>
<td>41.6</td>
<td>41.5</td>
</tr>
<tr>
<td>bbc_one_bbc_news</td>
<td>14.7</td>
<td>14.7</td>
<td>14.7</td>
</tr>
<tr>
<td>bbctwo_russia_a_journey_with_jonathan_dimbleby</td>
<td>23.0</td>
<td>23.2</td>
<td>23.2</td>
</tr>
<tr>
<td>bbctwo_graham_norton_uncut</td>
<td>44.6</td>
<td>44.4</td>
<td>44.6</td>
</tr>
</tbody>
</table>

Table A.2: % WERs of dev.full at show level: before and after adaptation using LHUC and Adapt-all approaches.
Bibliography


Peter Bell, Hitoshi Yamamoto, Pawel Swietojanski, Youzheng Wu, Fergus McInnes, Chiori Hori, and Steve Renals. A lecture transcription system combining neural network acoustic and language models. In INTERSPEECH, Lyon, France.


Bibliography


