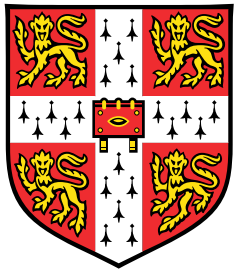
Scalable Recurrent Neural Network Language Models for Speech Recognition



Xie Chen

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Clare Hall College

March 2017

I would like to dedicate this thesis to my loving parents . . .

Declaration

This dissertation is the result of my own work carried out at the University of Cambridge and includes nothing which is the outcome of any work done in collaboration except where explicitly stated. It has not been submitted in whole or in part for a degree at any other university. Some of the work has been previously presented in international conferences [142, 40, 38, 41, 43, 44, 42, 45] and workshops, or published as a journal article [144, 46]. The length of this thesis including footnotes, appendices and references is approximately 63000 words. This thesis contains 48 figures and 44 tables.

Xie Chen
March 2017

Acknowledgements

First of all, I would like to express my utmost gratitude to my supervisor, Professor Mark Gales, for his mentorship and support over the past four years. I learned a lot from the regular discussion every week and his wisdom, insight, diligence and passion in research always encourage me. I believe this inspiration will be there with me for my whole life.

I am also in debt of Dr Xunying Liu, who brought me into the field of language models. He always stood by me throughout my work in recurrent neural network language models. I will miss the regular discussion every Wednesday.

Special thanks go to Toshiba Research Europe Ltd. and Cambridge Overseas Trusts for the financial support, allowing me to attend many international conferences and workshops.

I also want to thank my advisor, Professor Phil Woodland for his constructive suggestions. I owe my thanks to my colleagues in the Machine Intelligence Lab for the help and encouragement they have given to me. Particular thanks must go to Dr Kate Knill, Dr Anton Ragni, Dr Rogier Dalen, Dr Yu Wang, Dr Linlin Wang, Dr Yanmin Qian, Dr Yongqiang Wang, Dr Shixiong Zhang, Dr Pierre Lanchantin, Dr Penny Karanasou, Dr Jingzhou Yang, Dr Takuya Yoshioka, Chao Zhang, Chunyang Wu, Moquan Wan, Jeremy Wong, Andrey Malinin and Jake Vasilakes, for the scintillating discussions, whether it be speech recognition, machine learning, or subjects less directly related to our research. All have enriched my time here. I also would like to thank Patrick Gosling and Anna Langley for their excellent work in maintaining the computing facilities. These intelligent and kind people make the group an interesting place to work in.

The friends I met in Cambridge will be a treasure forever. The experience in the Chinese society also enriched my life and I have had the chance to make many great friends there. The accommodation I stayed in Cambridge for the last two years, HOA, has brought countless happy moments and make me feel at home. I would like to thank all people I met there.

I also want to thank Zhen Tong for her company and support, which means a lot to me. Finally, the biggest thanks go to my parents. For many years, they have offered everything possible to support me. Without their encouragements, I would not be here. This thesis is dedicated to them.

Abstract

Language Modelling is a crucial component in many areas and applications including automatic speech recognition (ASR). *n*-gram language models (LMs) have been the dominant technology during the last few decades, due to their easy implementation and good generalism on unseen data. However, there are two well known problems with *n*-gram LMs: data sparsity; and the *n*-order Markov assumption. Previous research has explored various options to mitigate these issues. Recently, recurrent neural network LMs (RNNLMs) have been found to offer a solution for both of these issues. The data sparsity issue is solved by projecting each word into a low, continuous, space, and the long term history is modelled via the recurrent connection between hidden and input layer. Hence, RNNLMs have become increasingly popular and promising results have been reported on a range of tasks. However, there are still several issues to be solved in area to apply RNNLMs to the ASR task. Due to the long term history, the training of RNNLMs is difficult to parallelise and slow to train on large quantities of training data and large model size. It is easy to apply Viterbi decoding or lattice rescoring for standard *n*-gram LMs as they have limited history, while it is difficult for RNNLMs because of their long term history. This thesis aims to facilitate the application of RNNLMs in ASR. First, efficient training and evaluation of RNNLMs are developed. By splicing multiple sentences, RNNLMs could be trained efficiently with bunch (i.e. mini-batch) mode on GPUs. Several improved training criteria are also investigated to further improve the efficiency of training and evaluation. Second, two algorithms are proposed for efficient lattice rescoring and compact lattices are able to generate. Third, the adaptation of RNNLMs is investigated. Model fine-tune and incorporation of informative feature based adaptation are investigated. Various topic models are applied to extract topic representation for efficient adaptation. Finally, the different modelling power of RNNLMs and *n*-gram LMs are explored and the interpolation of these two types of models is studied.

The first contribution of this thesis is the efficient training and inference of RNNLMs. The training of RNNLMs is computationally heavy due to the large output layer and difficulty of parallelisation. In most previous works, RNNLMs were trained on CPU with class based RNNLMs. In this thesis, a novel sentence splicing method is proposed, which allows RNNLMs to be trained much more efficiently with bunch mode. GPU is also used

to fully explore its parallelisation power for fast computation. In addition to the standard cross entropy based training criterion, two improved training criteria: variance regularisation and noise contrastive estimation, are studied for rapid RNNLM training and inference. Experiments show that significant speedup can be obtained for both training and testing.

The second contribution of this thesis is the lattice rescoring of RNNLMs. Due to the long term history, lattice rescoring of RNNLMs is difficult. Most previous work used N-best or prefix tree, which only rescore top N hypotheses using RNNLMs and can not generate compact lattices. In this thesis, we aims to apply RNNLMs for lattice rescoring. Approximations are made for RNNLM lattice rescoring to cluster similar histories. *n*-gram and history vector based clustering are proposed and used as criteria to cluster history and combine paths in lattices. Both of these two approaches are able to generate compact lattices, which can be used for applications including confusion network decoding and key word spotting with performance improvement.

The third contribution of this thesis is the study of efficient adaptation for RNNLMs. Two popular approaches for RNNLM adaptation: model fine-tuning and incorporation of informative feature, are investigated and compared. A range of topic models are used to extract topic representation for efficient adaptation. The experiments show that the unsupervised RNNLM adaptation yield significant perplexity reduction and moderate word error rate improvement on a large quantities of data compared to standard RNNLMs.

Another contribution of this thesis lies in the interpolation between *n*-gram LMs and RNNLMs. Based on an experimental analysis of interpolation between RNNLM and *n*-gram LM, back-off level is used as a feature to cluster and share parameters for interpolation. Two back-off based interpolation algorithms are proposed and investigated.

It is also worth mentioning that the work described in this thesis has been developed to an open source toolkit: CUED-RNNLM [41]. This toolkit supports efficient RNNLM training on GPU, evaluation on CPU, RNNLM lattice rescoring and adaptation. It has been used by a number of speech groups from universities and institutes.

Table of contents

List of figures **xv**

List of tables **xix**

Nomenclature **xxiii**

1 Introduction **1**
1.1 Overview of Automatic Speech Recognition Systems 2
1.2 Thesis Organisation 3
1.3 Contributions and Collaborations 4

2 Automatic Speech Recognition **7**
2.1 Front-End Feature 7
2.2 Hidden Markov Model based Acoustic Model 7
2.2.1 Gaussian Mixture Model 9
2.2.2 Deep Neural Network 12
2.2.3 Tandem System 13
2.2.4 Joint Decoding 15
2.2.5 Acoustic Model Adaptation 16
2.3 Language Model 17
2.3.1 *n*-gram Language Model 19
2.3.2 Smoothing 20
2.3.3 Language Model Interpolation 22
2.3.4 Improved *n*-gram Language Model 23
2.3.5 Beyond *n*-gram Language Model 25
2.3.6 Language Model Adaptation 26
2.4 Search 28
2.4.1 Decoding 29
2.4.2 Lattice Rescoring 31

2.4.3 Confusion Network Decoding 32
2.5 Evaluation 33
2.6 Summary 34

3 Neural Network Language Models 35
3.1 Model Structure 36
3.1.1 Feedforward Neural Network Language Model 36
3.1.2 Recurrent Neural Network Language Model 39
3.1.3 Long Short Term Memory based RNNLM 41
3.2 Training of Neural Network based Language Model 43
3.2.1 Cross Entropy 43
3.2.2 Back Propagation 43
3.2.3 Back Propagation Through Time 45
3.3 Application of RNNLMs for Speech Recognition 46
3.4 Summary 47

4 Efficient Training and Inference of RNNLMs 49
4.1 Recurrent Neural Network LMs Structures 50
4.1.1 Full output layer based RNNLMs (F-RNNLMs) 50
4.1.2 Class Based RNNLMs (C-RNNLMs) 51
4.2 RNNLM Training Criteria 53
4.2.1 Cross Entropy 53
4.2.2 Variance Regularisation 54
4.2.3 Noise Contrastive Estimation 55
4.3 Computation Complexity Analysis 58
4.4 Implementation 59
4.4.1 RNNLM training word by word 60
4.4.2 Conventional Bunch Mode RNNLM Training 61
4.4.3 Bunch Mode RNNLM Training with Sentence Splicing 62
4.4.4 Analysis of the Efficiency of Parallelisation 62
4.4.5 Efficient Softmax Calculation and Parameter Tuning 64
4.5 Pipelined Training of RNNLMs 65
4.6 Experiments 66
4.6.1 CTS-English ASR Experiment 66
4.6.2 Google's One Billion Word Experiment 72
4.7 Conclusions 74

5 Efficient Lattice Rescoring of RNNLMs 75
5.1 *n*-gram LM approximation of RNNLM . . . 76
5.2 N-best Rescoring of RNNLM . . . 77
5.3 History Combination of *n*-gram LM . . . 78
5.4 Lattice Rescoring of RNNLM . . . 79
5.4.1 *n*-gram based History Clustering . . . 80
5.4.2 History Vector Distance based Clustering . . . 81
5.4.3 Algorithm Implementation . . . 83
5.5 Experiments . . . 84
5.5.1 Experiments on English CTS Data . . . 84
5.5.2 Experiments on Mandarin CTS Data . . . 86
5.5.3 Experiments on Babel Corpus . . . 90
5.6 Summary . . . 96

6 Adaptation of RNNLMs 97
6.1 Review of RNNLM Adaptation . . . 97
6.1.1 Fine-tuning . . . 98
6.1.2 Incorporating Auxiliary Feature . . . 98
6.2 Adaptation using Genre Information . . . 100
6.3 Topic Space based Adaptation . . . 101
6.3.1 Probabilistic Latent Semantic Analysis . . . 102
6.3.2 Latent Dirichlet Allocation . . . 102
6.3.3 Hierarchical Dirichlet Process . . . 103
6.4 Experiments . . . 104
6.4.1 Experimental Setup . . . 104
6.4.2 Results for RNNLMs trained on 11M words . . . 105
6.4.3 Results for RNNLM trained with additional Subtitle Data . . . 106
6.5 Summary . . . 107

7 Interpolating RNNLMs and *n*-gram LMs 109
7.1 Linear Interpolation . . . 110
7.2 Back-off Based LM Interpolation . . . 110
7.2.1 Generalised LM Interpolation using Weight Clustering . . . 110
7.2.2 Interpolation using Back-off for Weight Clustering . . . 111
7.2.3 Back-off based Interpolation with Rescaling . . . 113
7.3 Experiments . . . 115
7.3.1 Experiment on Penn TreeBank Corpus . . . 115

7.3.2 Experiments on Babel Corpus 116
7.3.3 Experiments on MGB Corpus 117
7.4 Conclusion and Discussion 118

8 Experiments on Meeting Transcription 121
8.1 Data Description 121
8.1.1 AMI Corpus 121
8.1.2 Toshiba Technical Meeting Data 123
8.2 Baseline System 125
8.2.1 Acoustic Modelling 125
8.2.2 Language Modelling 126
8.2.3 Baseline Results 127
8.3 Experiments with RNNLMs 127
8.4 Summary 131

9 Conclusion and Future Work 133
9.1 Conclusion 133
9.2 Review of Work 133
9.3 Future Work 134

References 137

Appendix A Calculation of Gradient in Noise Contrastive Estimation 155

Appendix B Experiment on AMI IHM corpus 157

List of figures

1.1 *A framework of speech recognition system.* 3

2.1 A left-to-right HMM model with 3 emitting states 8
2.2 An illustration of deep neural network 13
2.3 An illustration of deep neural network 14
2.4 An illustration of joint decoding system for acoustic modelling 15
2.5 An illustration of deep neural network based acoustic model adaptation using i-vector 17
2.6 An illustration of back-off scheme in a trigram LM 20
2.7 A decomposition of sentence in speech recognition 30
2.8 *An example of lattice with reference "well I think that is true"* 32
2.9 *An example of confusion network for sentence with the reference "well I think that is true"* 33
2.10 An example of WER computation 34

3.1 Feedforward neural network language model 37
3.2 Recurrent neural network language model 40
3.3 *Long short-term memory unit. The gating functions, input, forget and output gates, are introduced into the model to control the signal flow.* 41
3.4 *LSTM based language model with one LSTM layer.* 42
3.5 Back propagation through time for Recurrent NNLM 46

4.1 *An example RNNLM with an full output layer, An out-of-vocabulary (OOV) node is added in the input layer and out-of-shortlist (OOS) nodes is added in the output layer to model unseen words.* 50
4.2 *An example RNNLM with a class-based output layer. An out-of-vocabulary (OOV) node is added in the input layer and out-of-shortlist (OOS) nodes is added in the output layer to model unseen words.* 52

4.3 *Noise samples generated for sentence "<s> eating at home </s>“ by a unigram noise model for NCE training.* 57

4.4 *RNNLM training in one sentence. The blue line shows the process of sentence by sentence update and the recurrent vectors are computed by the same model without update until seeing the complete sentence; the green and read line are for word by word update. The green line shows the update of word w_5 using approximate recurrent vectors generated by old model parameters. The red line shows the correct way for word by word update, all history recurrent vectors are re-computed after each update* 60

4.5 *An example of bunched RNNLM training without sentence splicing. NULL tokens are added in the end of sentences to get the same sentence length for all sentences* 61

4.6 *An example of bunched RNNLM training with sentence splicing. All sentences from the training corpus are concatenated into M long streams and NULL tokens are only needed at the end of the training corpus.* 63

4.7 *An example of M sentences with the same sentence length N.* 64

4.8 *An example of data flow in pipelined RNNLM training using 2 GPUs* 65

4.9 *F-RNNLM training speed with and without sentence splicing on the CTS task. The red line is the train speed with spliced sentence bunch training and the green line is without sentence bunch but only aligning multiple sentences.* 68

4.10 *Variance of the output layer log normalisation term $\overline{\ln Z}$ on the validation data on CTS task at different epochs during NCE based RNNLM training.* . 71

5.1 *Example lattice for rescoring* 77

5.2 *Example N-best list for rescoring* 77

5.3 *Example prefix tree for rescoring* 78

5.4 *An example of combining two paths ending with (w_{i-1}, w_i) when 2-gram LM is used. S_A and S_B are the history states (e.g. acoustic model and language model likelihood, valid n-gram history) of these two paths.* 79

5.5 *An example of n-gram based history clustering. For the node with word “a”, its two history paths “<s> there is” and “<s> here is” has the same recent word “is”. When 2-gram approximation is applied, these two paths can be merged and only the history path (“<s> there is”) giving higher probability is kept and will be used for computing new history vector.* 81

5.6 *An example of history vector distance based clustering. For the node with word "a", its two history paths "<s> there is" and "<s> here is" can be represented by two history vectors in RNNLMs. These two paths can be merged as the Euclidean distance of these two vectors is smaller than a threshold. The history path ("<s> there is") giving higher probability is kept and will be used for computing new history vector.* 82
5.7 *WER and lattice density for n-gram approximation based RNNLM lattice rescoring on the English CTS task on* **eval04** 87
5.8 *WER and lattice density for history vector distance based RNNLM lattice rescoring on the English CTS task on* **eval04** 88
5.9 *WER and lattice density for n-gram approximation based RNNLM lattice rescoring on the Mandarin CTS* **eval97**. 91
5.10 *WER and lattice density for history vector distance based RNNLM lattice rescoring on the Mandarin CTS* **eval97**. 92
5.11 *A framework of keyword search system.* 92

6.1 *A standard framework for unsupervised RNNLM adaptation in speech recognition.* 98
6.2 *RNNLM adaptation with genre information based on fine-tuning.* 99
6.3 *An example RNNLM with an additional input feature vector* $\boldsymbol{f}$. 100

7.1 *n-gram dependent interpolation of n-gram LM and RNNLM.* 112
7.2 *An illustration of back-off scheme in a trigram LM.* 112

8.1 *Toshiba Technical Meeting Recording Configuration* 123

List of tables

4.1 Computational complexities for each RNNLM forward in the output layer during training and testing using different model structures and training criteria. 59

4.2 The empirical value for initial learning rate per sample with different bunch size for RNNLM training. 65

4.3 Training speed, perplexity and WER results of CPU trained C-RNNLMs on the CTS task with varying hidden nodes with cross entropy based training. . 67

4.4 Training speed, perplexity and WER results of GPU trained F-RNNLMs on **dev04** with varying bunch sizes and a fixed hidden layer size of 500. 68

4.5 Evaluation speed of C-RNNLMs (class based RNNLMs) and F-RNNLMs (full output layer RNNLMs) for N-Best scoring on **dev04**. The F-RNNLM is trained with bunch size 128. 69

4.6 Perplexity and WER performance of F-RNNLMs trained with variance regularisation on **dev04**. The mean and variance of log normalisation term were computed over the validation data. The two columns under WER (Z(h) and Z) denote word error rates using normalised or approximated RNNLM probabilities computed using Equations 4.6 and 4.12. 70

4.7 Training and evaluation speed of F-RNNLMs trained with variance regularisation on the CTS task. C-RNNLMs were trained on CPU and F-RNNLMs on GPU. Both were evaluated on CPU. 70

4.8 Perplexity and WER performance, training and evaluation speed of NCE trained F-RNNLMs on the CTS task. C-RNNLMs trained on CPU and F-RNNLMs on GPU. Both evaluated on CPU. 72

4.9 The training speeds (w/s) using cross entropy (CE) and noise contrastive estimation (NCE) with different output layer size for F-RNNLMs on CTS task 72

4.10 Training speed, perplexity and WER performance of F-RNNLMs on **dev04** using pipelined CE training on CTS task. 72

4.11 Perplexity performance of RNNLMs on Google's one billion (for ASR (provided by Cantab Research) corpus) word corpus. 73
4.12 Perplexity performance of RNNLMs on Google's one billion word corpus using 793K vocabulary. The train is run on GPU and test is on CPU. 74

5.1 Performance of 4-gram LM approximation of RNNLM by sampling and N-best rescoring on the English CTS **dev04** task 85
5.2 Performance of RNNLM lattice rescoring using *n*-gram based history and history vector distance based clustering on **dev04** set 86
5.3 Performance of N-best rescoring and sampling with RNNLM on the English CTS task on **eval04** 87
5.4 Performance of N-best rescoring and sampling with RNNLM on Mandarin CTS **dev14** testset 89
5.5 Performance of RNNLM lattice rescoring using *n*-gram based history and history vector distance based clustering on Mandarin CTS **dev14** testset. . . 90
5.6 Performance of N-best rescoring and sampling with RNNLM on the Mandarin CTS **eval97** testset. 91
5.7 Statistics and perplexity results for 5 languages in the Babel corpora. 94
5.8 WER results of RNNLM on Pashto. 94
5.9 WER and KWS results of RNNLM trained with various criteria on Pashto. . 95
5.10 WER and KWS results of RNNLM trained on different languages. 96

6.1 Statistics of the BBC training and test data. 104
6.2 PPL and WER results for genre dependent RNNLMs on 1 week BBC test data. The genre dependent RNNLMs were constructed based on the well-trained genre independent RNNLM. Tandem-SAT system was used as acoustic model. 105
6.3 PPL and WER results for RNNLMs with various topic representation on 1 week BBC test data. The RNNLMs with topic representation were trained from scratch. The number of topics is set to 30 for all topic models. The hypotheses were obtained using the 4-gram LM. 106
6.4 PPL and WER results for RNNLM adaptation with LDA using different numbers of topics on 1 week BBC test data 106
6.5 PPL and WER on 1 week BBC test data using RNNLM trained on additional subtitle data. 107

7.1 Perplexity performance of baseline 5-gram LM, RNNLM and their linear interpolation over varying back-off *n*-gram orders on PTB test set. 113

7.2 Statistics of the three corpora used for experiments 115
7.3 PPL results on test set of PTB corpus. 116
7.4 PPL and WER results on Swahili for Babel 117
7.5 PPL and WER results on MGB task 118

8.1 Summary of the AMI Test Data 122
8.2 Summary of the Meeting Training Data 122
8.3 Summary of the Toshiba Technical Meeting (TTM) Data 124
8.4 Microphone distance and WER results of manual transcription and ASR result compared to “gold-standard” on meeting A0001 124
8.5 Statistics of the train corpora for the language model. The linear interpolation weights on each corpus were optimised on the AMI dev set. 126
8.6 Out of Vocabulary (OOV) % for AMI and TTM test data 127
8.7 WERs of baseline ASR system on AMI and TTM test sets with a 3-gram language model 127
8.8 The perplexity for AMI and TTM test data using various language models. The feedforward and recurrent NNLMs were trained on 2M acoustic transcription. 128
8.9 The WERs for AMI and TTM test data using various language models. The feedforward and recurrent NNLMs were trained on 2M acoustic transcription. 129
8.10 PPL and WER results of C-RNNLMs (class based RNNLMs) and F-RNNLMs (full output layer RNNLMs) with N-best and lattice rescoring on AMI corpus. The RNNLMs are trained on AMI acoustic model transcription only, which is about 2M and cross entropy is used for RNNLM training. The WER results of Viterbi decoding is presented in the table 129
8.11 WERs on AMI and TTM test data using 2M transcription using LDA based RNNLM adaptation 130
8.12 PPL and WER results on AMI and TTM test sets with RNNLMs trained on different amounts of training data. 130

B.1 WER results of AMI eval set using HTK and Kaldi Toolkits. A 4-gram language model was used and CN (HTK), MBR (Kaldi) decoding were applied. 157

Nomenclature

General Notation

s	a scalar is denoted by a plain lowercase letter
$\mathbf{v}$	a column vector is denoted by a bold lowercase letter
$\mathbf{X}$	a matrix is denoted by a bold uppercase letter
$p(\cdot)$	probability density function
$P(\cdot)$	probability mass distribution
$\mathcal{W}$	word sentence
$\hat{\mathcal{W}}$	optimal word sentence
$\mathbf{O}$	obervation sequence
$\mathbf{o}$	acoustic feature vector
$\triangle\,\mathbf{o}$	delta acoustic feature vector
π	initial state distribution
$\mathcal{A}$	state transiotion probability matrix
$\mathcal{B}$	state output probability distribution
$\mathcal{M}$	HMM model parameters
$\mathcal{F}$	objective function for the training of acoustic model
$\boldsymbol{\theta}$	Language model parameters
$J(\cdot)$	objective function for the training of language model

$\boldsymbol{\mu}$	mean vector in the Gaussian distribution
$\boldsymbol{\Sigma}$	covariance matrix in the Gaussian distribution
ω	mixture weight of component in the GMM model
S	state sequence
Q	phone sequence
$\mathcal{Q}(\cdot,\cdot)$	auxiliary function for expectation maximum
$\xi_{ij}(t)$	probability of being in state s_i at time t and state s_j at time $t+1$
$\gamma_j(t)$	posterior probability of being state s_j in at time t
$\alpha_j(t)$	joint likelihood of the partial observation sequence up to t
$\beta_j(t)$	likelihood of the partial observation sequence from time instance $t+1$ to the end
λ	interpolation weight for language model interpolation
$\mathcal{V}^{in}$	input layer vocabulary in the RNNLM
$\mathcal{V}^{out}$	output layer vocabulary in the RNNLM
$\phi^{in}(w)$	index of word w in the input layer
$\phi^{out}(w)$	index of word w in the output layer
$\mathbf{r}^{(w)}$	the 1-of-K coding vector for word w
$\mathbf{A}$	hidden layer matrix in RNNLM
$\mathbf{B}$	recurrent layer matrix in RNNLM
$\mathbf{U}$	output layer matrix in RNNLM
$\boldsymbol{\theta}$	model parameters in RNNLM
$J^{\mathsf{CE}}(\cdot)$	objective function of cross entropy
$J^{\mathsf{NCE}}(\cdot)$	objective function of noise contranstive estimation
$J^{\mathsf{VR}}(\cdot)$	the objective function of variance regularization
$Z(h)$	normalisation term for softmax function in the output layer for history h in RNNLM

Z constant normalisation term used in variance regularization and noise contranstive estimation

$\overline{\ln Z}$ log normalisation term

$D(\cdot,\cdot)$ normalised Euclidean distance for two history hidden vectors

$\Phi_{NG}(w_1^{i-1})$ history clustering in n-gram LMs

$\Phi_{RNN}(w_1^{i-1})$ history clustering in RNNLMs

d one document

D the set of document

z one topic

$\mathbf{T}$ the set of latent topics

$\mathbf{f}$ a vector of posterior probabilities among topics

$\mathbf{M}_T$ topic model

$\hat{\mathbf{M}}_T$ optimal topic model

$n(d_i, w_j)$ count of word w_j occurring in document d_i

$\psi^{(k)}_{w_0^{i-1}}$ the set of word w_i whose back-off level equal to k, given history w_1^{i-1}

$\Gamma(w_0, w_1^{i-1})$ back-off level for word w_i given history w_1^{i-1} in the n-gram LMs

$\lambda^{(NG)}$ interpolation weight for the n-gram LMs

$\lambda^{(RNN)}$ interpolation weight of RNNLM

Acronyms / Abbreviations

BPTT Back Propagation Through Time

ASR Automatic Speech Recognition

HMM Hidden Markov Model

AMI Augmented Multi-party Interaction

AM Acoustic Model

LM Language Model

CMLLR Constrained Maximum Likelihood Linear

CN Confusion Network

CTS Conversational Telephone Speech

CE Cross Entropy

CMN Cepstral Mean Normalisation

C-RNNLMs Class based Recurrent Neural Network Language Models

DNN Deep Neural Network

CVN Cepstral Variance Normalisation

FLP Full Language Package

F-RNNLMs Full output layer Recurrent Neural Network Language Models

EM Expectation Maximisation

GPU Graphics Processing Units

GMM Gaussian Mixture Model

HTK Hidden Markov Model Toolkit

HDP Hierarchical Dirichlet Processes

HLDA Heteroscedastic Linear Discriminant Analysis

LDA Latent Dirichlet Allocation

LSTM Long Short Term Memory

LVCSR Large Vocabulary Continuous Speech Recognition

MAP Maximum A Posterior

MBR Minimum Bayesian Risk

MCE Minimum Classfication Error

MGB Multi-Genre Broadcast

MFCC Mel-Frequency Cepstral Coeficients

ML Maximum Likelihood

MLLR Maximum Likelihood Linear Regression

MMI Maximum Mutual Information

MTWV Maximum Term-Weighted Value

NCE Noise Contrastive Estimation

MPE Minimum Phone Error

PPL Perplexity

OOV Out of Vocabulary

RNN Recurrent Neural Network

PLSA Probabilistic Latent Semantic Analysis

PLP Perceptual Linear Prediction

SAT Speaker Adaptive Training

TTM Toshiba Technical Meeting

SI Speaker Independent

SGD Stochastic Gradient Descent

SD Speaker Dependent

WER Word Error Rate

VR Variance Regularisation

Chapter 1

Introduction

Speech is one of the most natural ways to communicate between people. It plays an important role in our daily lives. To make machines able to talk with people is a challenging but very useful task. A crucial step is to enable machines to recognise and understand what people are saying. Hence, speech recognition becomes a key technique providing an interface for communication between machines and humans. There has been a long research history on speech recognition [102].

The first speech recognition system, a digit recogniser, was invented in 1952 in Bell lab [53]. Since then, research on speech recognition has been carried out in both academia and industry and gained vast attention [102]. Hidden Markov models (HMMs) [9, 109] were introduced into speech recognition in the 1970s, and became the cornerstone in the area of speech recognition. The standard HMM has been refined in a number of ways such as state clustering, adaptation and discriminative training in subsequent years [71]. Significant progress has been achieved in speech recognition over the last several decades. In the early years, speech recognition was studied on isolated word recognition, with small vocabulary size (e.g. several hundreds). Native speakers spoke under clean environment, such as reading speech. Nowadays, large vocabulary (hundreds of thousands of words), continues speech recognition becomes the main research interest, such as voice search [31] and conversational telephone speech recognition [194]. The speech is spontaneous under diverse acoustic environments, which is more similar to how people behave in their daily lives. Advance in computer hardware (multi-core CPU and GPU) and parallel algorithms also facilitate the use of dramatically increasing amount of training data. Nowadays, thousands of hours of speech and billions of text data can be used to train recognition systems. Various adaptation techniques [71] are also developed to address the acoustic mismatch caused by speaker, noise, channel and so on. The improvement in performance can be also obtained from multi-microphone by overcoming reverberation and reducing noise [249]. Recently, deep learning has attracted extensive research interests and presented significant improvement in performance over a range of tasks [97]. With the big advance in speech recognition techniques, many companies have integrated speech recognition into products, such as Siri from Apple, Google watch from Google and speech translation in Skype from Microsoft. It is clear that the speech recognition techniques are entering our daily lives and gradually changing the way of life.

1.1 Overview of Automatic Speech Recognition Systems

Modern ASR systems are mainly based on statistical approaches under a Bayesian framework [71]. Mathematically, given the obervation $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, ...\mathbf{o}_T\}$ (i.e. feature extracted from raw speech signal), the probability of the specific word sequence $\mathcal{W} = \{w_1, w_2, ..., w_N\}$ can be written as $P(\mathcal{W}|\mathbf{O})$. According to Bayesian decision rules, the most likely word sentence $\hat{\mathcal{W}}$ can be obtained as,

$$\hat{\mathcal{W}} = \arg\max_{\mathcal{W}} P(\mathcal{W}|\mathbf{O}) \tag{1.1}$$

This can be rewritten based on Bayes' formula,

$$\begin{aligned}\hat{\mathcal{W}} &= \arg\max_{\mathcal{W}} P(\mathcal{W}|\mathbf{O}) \\ &= \arg\max_{\mathcal{W}} \frac{p(\mathbf{O}|\mathcal{W})p(\mathcal{W})}{P(\mathbf{O})} \\ &= \arg\max_{\mathcal{W}} p(\mathbf{O}|\mathcal{W})P(\mathcal{W})\end{aligned} \tag{1.2}$$

where the probability of the observation $p(\mathbf{O})$ can be omitted since it is independent of the word sequence $\mathcal{W}$. The posterior probability $P(\mathcal{W}|\mathbf{O})$ can be split into two components in the Equation 1.2: $p(\mathbf{O}|\mathcal{W})$ and $P(\mathcal{W})$. $p(\mathbf{O}|\mathcal{W})$ is the likelihood of observation $\mathbf{O}$ given word sequence $\mathcal{W}$, which is called the acoustic model in the literature. $P(\mathcal{W})$ is the prior probability of the word sequence $\mathcal{W}$, which is called the language model. The acoustic model is usually trained on audio corpus where the speech and its word sequence label (i.e. transcription) are given, and language model is trained on text corpus where a large number of word sequences are available. Given the acoustic model and language model, the posterior probability of a specific word sequence $\mathcal{W}$ can be calculated. The word sequence with the highest posterior probability is chosen as the recognition result as shown in Equation 1.2.

Figure 1.1 shows a standard framework of ASR system. The acoustic and language models are prepared before recognition. The lexicon (also known as pronunciations dictionary) specifies the pronunciation of each word in the vocabulary. The pronunciation of a word can be generated manually by experts or automatically by grapheme to phoneme (g2p) algorithms [17]. The information from the acoustic model, language model and lexicon are integrated in a decoder. For each utterance to be recognised, acoustic features can be extracted from the raw speech waveform using front-end processing [210]. The decoder takes the acoustic feature as input, searches from the search space constraint by acoustic model, language model and lexicon, and finally generates the most possible hypothesis based on Equation 1.2.

This thesis mainly focuses on the language model. The language model aims to model the probability of any given word sequence. A range of language models have been proposed for speech recognition. *n*-gram language models (*n*-gram LMs) are the most popular language model and have been the dominating language model during the last several decades. More recently, recurrent neural network language models (RNNLMs) have shown

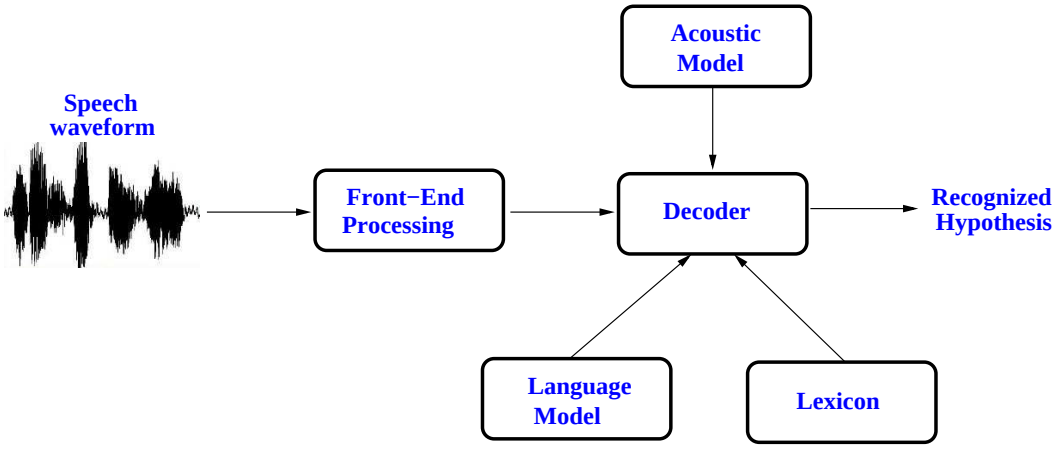


Fig. 1.1 *A framework of speech recognition system.*

promising performance in a range of applications including speech recognition [154, 61, 216, 55, 123]. However, some issues still exist when RNNLMs are applied in speech recognition, such as long training time for large training corpora [39] and difficulty of lattice rescoring [142]. In this thesis, the application of RNNLMs in state-of-the-art speech recognition systems is studied.

1.2 Thesis Organisation

This thesis is organised as follows,

Chapter 2 introduces the key techniques in speech recognition systems. The extraction of acoustic feature from speech signal, training of the acoustic model and language model, as well as the search (i.e. decoding) are described.

The neural network based language model is the research topic in this thesis. Chapter 3 reviews three popular types of neural network based language model, including feedforward, recurrent and LSTM based neural network language model.

The efficient RNNLM training and inference are explored in Chapter 4. The training of RNNLMs is computationally heavy due to the large output layer and difficulty of parallelisation. A novel sentence splicing method is proposed, which allows RNNLMs to be trained much more efficiently with bunch mode. GPU is also used to fully explore its parallelisation power for fast computation. In addition to the standard cross entropy based training criterion, two improved training criteria: variance regularisation and noise contrastive estimation, are studied for rapid RNNLM training and inference.

Latticie rescoring using RNNLMs is studied in Chapter 5. Due to the long term history, lattice rescoring of RNNLMs is difficult. Most previous work used N-best or prefix tree, which only rescore top N hypotheses using RNNLM and can not generate compact lattices. Approximations are made for RNNLM lattice rescoring to cluster similar histories in this thesis. *n*-gram and recurrent vector based clustering are proposed and used as criteria to cluster history and combine paths in lattices. Both of these two approaches are able to

generate compact lattices, which can be used for applications including confusion network decoding and key word spotting.

Chapter 6 studies the adaptation of RNNLMs. Two popular approaches for RNNLM adaptation: model fine-tuning and incorporation of informative feature, are investigated and compared. A range of topic models are used to extract topic representation for efficient adaptation.

Chapter 7 investigates the interpolation between RNNLMs and *n*-gram LMs. Based on an experimental analysis of interpolation between RNNLM and *n*-gram LM, back-off level is used as a feature to cluster and share parameters for interpolation. Two back-off based interpolation algorithms are proposed and investigated.

Chapter 8 examines the techniques discussed in the thesis on a meeting transcription task. Several sources of public meeting data including AMI, ICSI and NIST meetings are used for training, and two sets of test data are used. The first one is from the standard AMI test data and the other test set is a series of real meetings collected by Toshiba.

Finally, this thesis concludes in Chapter 9 with a summary of contribution and a discussion of future work.

1.3 Contributions and Collaborations

This thesis covers a wide range of topics concerning RNNLMs for speech recognition. This thesis has benefited from the help of various collaborators. In this section, I will give a brief summary of contributions made by these collaborators. Prof. Mark Gales is my supervisor and he has involved in idea discussion, experiments and paper writting for all papers associated with this thesis.

(a) The work on efficient RNNLM training and evaluation in Chapter 4 has been published in 3 conference papers and 1 journal article [40, 43, 44, 46]. I was responsible for the original ideas, code implementation, experiments and paper writting. The collaborators provided baseline ASR systems and helped on paper correction. Yongqiang had been attending most of the related discussions till he joined Microsoft in 2014.

(b) The work on RNNLM lattice rescoring in Chapter 5 has been published 2 conference papers and 1 journal article [142, 47, 144]. The original ideas came from weekly discussions between Xunying, Yongqiang and I. We have equal contributions on the idea. Xunying was the person who implemented the lattice expansion code in HLRescore (HTK [29]) and provided all baseline ASR systems. I was responsible for the code related to RNNLM training, model input, probability computation and experiments. Anton Ragni and Jake Vasilakes provided baseline ASR systems for keyword search. [142, 144] were written by Xunying Liu and I wrote [47].

(c) The work on RNNLM adaptation in Chapter 6 has been published in 1 conference paper [45]. I contributed the original idea and was responsible for implementing RNNLM adaptation code, experiments and paper writting. The collaborators helped to train PLSA models and extract PLSA features; provided baseline ASR systems, and extracted HDP features.

(d) The work on language model interpolation in Chapter 7 has been published in 1 conference paper [42]. I contributed the original ideas, was responsible for implementation of the RNNLM interpolation code, experiments and paper writting. The collaborators provided baseline ASR systems and helped on paper correction.

Chapter 2

Automatic Speech Recognition

This chapter describes key components of the standard automatic speech recognition (ASR) system. As shown in Figure 1.1, these include acoustic feature extraction, acoustic model, language model and decoder.

2.1 Front-End Feature

In speech recognition, acoustic features are extracted from the raw speech signal. The acoustic features are expected to carry sufficient information from speech as well as to be a suitable form for modelling. Mel frequency cepstral coefficient (MFCC) [148, 54], perceptual linear predictive (PLP) [94], filter bank (FBANK) [160], Tandem [95] are popular acoustic feature.

2.2 Hidden Markov Model based Acoustic Model

Hidden Markov Model (HMM) has been the most popular model for speech recognition systems since the 1980s [181, 180]. HMM provides an elegant way to model the continuous and dynamic character in speech signal. ASR systems considered in this thesis are based on the Hidden Markov Model (HMM) [1]. In this section, the basic concept of HMM will be presented.

Two assumptions are made to allow HMM to be suitable for modelling speech.

- **Quasi-stationarity:** Speech can be split into short segments or states, in which the speech signal is stationary. The transition between states is instantaneous.
- **Conditional independence:** The acoustic feature vector is only dependent on the current state. Given the state, the feature vector is conditionally independent from the previous or following feature vectors. And the transition probability of the next state depends only on the current state, irrelevant of the feature vectors.

[1] The recent proposed alternatives of HMM to model the speech signal based on LSTM-CTC models [82, 80, 192] and attention model [28] are not discussed in this thesis.

Neither of these assumptions is true for speech signal in practice. The speech signal is not stationary and varies quickly with time. The successive feature vectors are also highly correlated. Much works has been carried out to alleviate these assumptions. However, HMM is still a successful model and gives good performance for speech recognition under these assumptions.

Figure 2.1 shows a typical left-to-right HMM model with five states. It is a finite state machine with one entry state 1, one exit state 5 and three emitting states. The entry state 1 and exit state 5 are non-emitting states. The emitting states 2, 3 and 4 allow self-loop transitions. The acoustic feature vector $\mathbf{o}_t$ is generated at each time stamp t. Its probability density function (PDF) depends on the current state; e.g., when the current state is 2, the PDF of $\mathbf{o}_t$ is $b_2(\mathbf{o}_t)$. In the next time step, the state is 2 with probability a_{22} and 3 with probability a_{23}. In this way, a variable length speech signal can be modelled by the HMM.

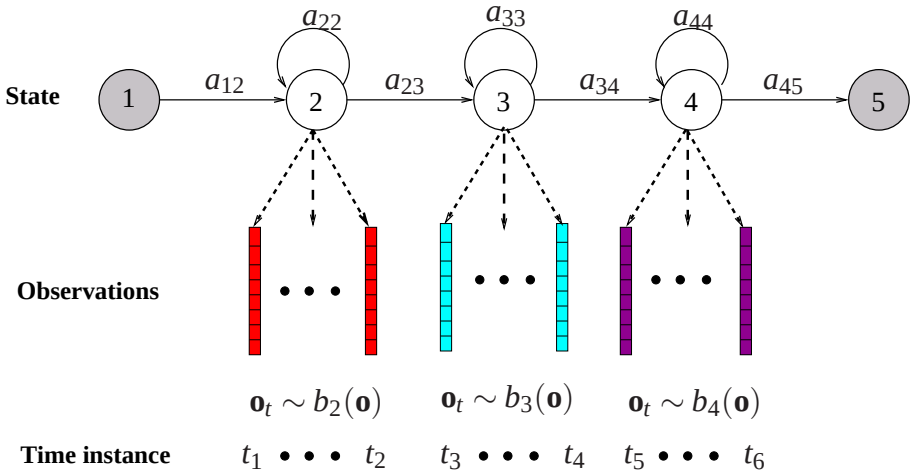


Fig. 2.1 *A left-to-right HMM model with 3 emitting states, states 1 and 5 are non-emitting states and 2,3,4 are emitting states*

There are several parameters to be estimated.

1 π: initial state distribution
The initial state probability is,

$$\pi_i = P(\phi_0 = s_i) \tag{2.1}$$

where ϕ_t denotes the state at time t. For $\Pi = \{\pi_1, \pi_2, ...\pi_N\}$ to be a valid distribution, it must satisfy,

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.2}$$

where N is the number of states. For the left-to-right HMM shown in Figure 2.1, the initial state is state 2 with probability 1.

2 $\mathcal{A}$: state transition probability matrix
$\mathcal{A} = \{a_{ij}\}$ defines transition probabilities from state s_i to state s_j

$$a_{ij} = P(\phi_t = s_j | \phi_{t-1} = s_i) \tag{2.3}$$

which satisfies the following constraints if $\mathcal{A}$ is a full matrix, transitions between each pair of states are allowed.

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{2.4}$$

In many applications, $\mathcal{A}$ does not need to be full. For a left-to-right HMM shown in Figure 2.1, the transition probability matrix $\mathcal{A}$ is written as,

	s_2	s_3	s_4	s_5
s_2	a_{22}	a_{23}	0	0
s_3	0	a_{33}	a_{34}	0
s_4	0	0	a_{44}	a_{45}

where each state is constrained to jump to itself or the next state. s_1 and s_5 are non-emitting states, they will jump to the next state immediately.

3 $\mathcal{B}$: state output probability distribution
$\mathcal{B} = \{b_1(\mathbf{o}_t), b_2(\mathbf{o}_t), ..., b_N(\mathbf{o}_t)\}$ are PDFs of acoustic feature vectors given states. The PDF of acoustic feature vector $\mathbf{o}_t$ in state s_i at time t can be written as,

$$b_i(\mathbf{o}_t) = p(\mathbf{o}_t | \phi_t = s_i) \tag{2.5}$$

The PDF $b_i(\mathbf{o}_t)$ is also called a likelihood. There are mainly two popular acoustic models to compute likelihoods in ASR systems. One is a Gaussian Mixture Model (GMM). The GMM-HMM system is constructed using this form of PDF. The other is a Deep Neural Network (DNN) [97], which yields a so-called DNN-HMM system (also known as a Hybrid system) [199]. The GMM and DNN will be introduced in Chapters 2.2.1 and 2.2.2 respectively. There are also other variants such as recurrent neural network for acoustic modelling [80]. They are out of the scope of this thesis.

2.2.1 Gaussian Mixture Model

Gaussian Mixture Model (GMM) is a mixture of finite multivariate Gaussian distribution. Theoretically, GMM is able to approximate any distribution when there are sufficient Gaussian components. GMM is widely used for modelling state emission probabilities in speech recognition. Given state s_i, the output PDF of observation $\mathbf{o}_t$ is computed as below,

$$p(\mathbf{o}_t | \Lambda_i) = \sum_{k=1}^{K} \omega_{ik} \mathcal{N}(\mathbf{o}_t; \boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik}), \tag{2.6}$$

where state parameters $\Lambda = \{\Lambda_i\} = \{\omega_{ik}, \boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik}\}$ consist of mixture weights $\{\omega_{ik}\}$, mean vectors $\{\boldsymbol{\mu}_{ik}\}$ and covariance matrices $\{\boldsymbol{\Sigma}_{ik}\}$. The mixture weights for each state s_i must satisfy the following constraints:

$$\sum_{k=1}^{K} \omega_{ik} = 1 \quad and \quad \omega_{ik} > 0 \tag{2.7}$$

The multivariate Gaussian distribution for each component can be written as,

$$\mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{d}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(\mathbf{o}_t - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o}_t - \boldsymbol{\mu})\} \tag{2.8}$$

where d is the dimension of feature vector. Typically, the diagonal covariance matrices are used. To reduce the number of parameters for estimation and simplify the computation, there have been also efforts on incorporating more power forms [7, 69]. In this work, we only consider the diagonal covariance matrix for GMM.

Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is one of the most popular training criteria for HMM parameter estimation. The aim is to find the optimal model parameter to maximize the observation likelihood given the reference transcription. The objective function of MLE can be written as,

$$\mathcal{F}_{ML}(\mathcal{M}) = \log(p(\mathbf{O}|\mathcal{W};\mathcal{M})) \tag{2.9}$$

where $\mathcal{W}$ is a word transcription for acoustic observations $\mathbf{O}$. The ML objective function can be rewritten as,

$$\begin{aligned}
\mathcal{F}_{ML}(\mathcal{M}) &= \log(p(\mathbf{O}|\mathcal{W};\mathcal{M})) \\
&= \log\Big\{\sum_{S,Q} p(\mathbf{O},S|Q;\mathcal{M})p(Q|\mathcal{W})\Big\} \\
&= \log\Big\{\sum_{S,Q}\big(a_{\phi_0\phi_1}\prod_{t=1}^{T} p(\mathbf{o}_t|\phi_t;\mathcal{M})a_{\phi_t\phi_{t+1}}\big)p(Q|\mathcal{W})\Big\}
\end{aligned} \tag{2.10}$$

where t is frame index and ϕ_t is the state index at time t. $p(\mathbf{o}_t|\phi_t;\mathcal{M})$ is the GMM likelihood as shown in Equation 2.6. Q is a particular phone sequence corresponding to the word sequence $\mathcal{W}$. Although it is possible that there are multiple pronunciations, usually only the highest likely pronunciation is used for MLE estimation. Hence, the sum over Q in Equation 2.10 can be removed. S is any possible state alignment given the phone sequence derived from the word sequence. The length of the state sequence is the same as that of the observation sequence $\mathbf{O}$. Hence, the objective function of Equation 2.10 can be simplified as

$$\mathcal{F}_{ML}(\mathcal{M}) = \log\Big\{\sum_{S} p(\mathbf{O},S|\mathcal{W};\mathcal{M})\Big\} \tag{2.11}$$

There are two hidden variables during the above estimation, the state ϕ_t and the GMM component ω_i in each state. Expectation Maximum (EM) algorithm can be used to optimise Equation 2.11 [16].

Discriminative Training

Maximum Likelihood is the optimal criterion when two conditions are satisfied, i.e. sufficient training data and correct model assumption. However, neither of these two conditions is satisfied. Hence, discriminative techniques were introduced into acoustic model training as these do not assume infinite amount of data or the correct model. Rather than maximising the likelihood of observation given the correct transcription in MLE training, discriminative training aims to maximise the posterior probability given the observation. The recognition accuracy metric can also be taken into account. There are a range of discriminative criteria proposed for speech recognition. Significant and consistent performance improvements were reported in various tasks [178]. Hence, discriminative training is widely used in the state-of-the-art speech recognition systems. In this chapter, several common discriminative criteria are reviewed briefly, including maximum mutual information (MMI) and minimum Bayers' risk (MBR).

Maximum Mutual Information

MMI [8, 178] training attempts to maximum the mutual information between reference sentence $\mathcal{W}$ and observation sequence $\mathbf{O}$. The objective function can be written as below,

$$\begin{aligned}\mathcal{F}_{MMI}(\mathcal{M}) &= I(\mathcal{W},\mathbf{O}|\mathcal{M}) \\ &= \log\big(\frac{p(\mathcal{W},\mathbf{O}|\mathcal{M})}{P(\mathcal{W})p(\mathbf{O}|\mathcal{M})}\big)\end{aligned} \tag{2.12}$$

Normally the language model probability $P(\mathcal{W})$ is not jointly trained with the acoustic model since the language model is trained on a far larger corpus. Given that $P(\mathcal{W})$ is fixed, MMI objective function is equivalent to maximising the average log-posterior probability for the reference sentence $\log P(\mathcal{W}|\mathbf{O},\mathcal{M})$, which is normally written as below,

$$\mathcal{F}_{MMI}(\mathcal{M}) = \log\big(\frac{p(\mathbf{O}|\mathcal{W},\mathcal{M})^{\frac{1}{k}}P(\mathcal{W})}{\sum\limits_{\mathcal{W}'}p(\mathbf{O}|\mathcal{W}',\mathcal{M})^{\frac{1}{k}}P(\mathcal{W}')}\big) \tag{2.13}$$

The denominator term is the probability of observation sequences $p(\mathbf{O}|\mathcal{M})$, which sums over all possible word sequences $\mathcal{W}'$. In practice, a lattice containing highest probable word sequences as used to approximate all possible word sequences. k is the language model scale factor, which aims to scale down the dynamic range of acoustic score to yield a broader posterior probability distribution.

Minimum Bayes's Risk

The Minimum Bayes' Risk (MBR) aims to minimise the expected loss. The expected loss can be expressed as,

$$\mathcal{F}_{MBR}(\mathcal{M}) = \sum_{W'} P(\mathcal{W}|\mathbf{O};\mathcal{M})L(\mathcal{W}',\mathcal{W}) \tag{2.14}$$

where $L(\mathcal{W}',\mathcal{W})$ is the loss function for the output hypothesis $\mathcal{W}$ given the reference word sequence $\mathcal{W}$. A number of loss functions can be found in the literature. These differ in terms of minimising error at different levels such as sentence, word and phone.

- Sentence: this form is shown in Equation 2.15. It aims to minimise error at the sentence level.

$$L_{\mathcal{W}',\mathcal{W}} = \begin{cases} 1 & if \quad \mathcal{W}' \neq \mathcal{W} \\ 0 & if \quad \mathcal{W}' = \mathcal{W} \end{cases} \tag{2.15}$$

- Word: The loss function can be defined at a word level. It leads to criterion as called minimum word error (MWE) rate.

- Phone: When phone is selected as the unit to compute loss function, it results in a minimum phone error (MPE) criterion, which is applied widely in large vocabulary speech recognition system [178].

2.2.2 Deep Neural Network

An alternative state output distribution can be obtained using a neural network. This approach can be traced back to 1980s. Feedforwad Neural network and recurrent Neural network were proposed to substitute GMM based acoustic modelling [183, 19, 186, 21], where a single hidden layer was normally used and monophone was chosen as the target during training. However, due to their long train time and difficulty of adaptation, GMM-HMM has been the state-of-the-art for the last two decades. Neural network revives with the advance of deep neural network containing many hidden layers since 2006. [96] proposed a novel pretraining method to construct a deep neural network and obtained significant improvement on image task. This model was introduced into speech recognition in [51] and promising results were reported on a range of LVCSR systems [199, 97]. The system using DNN to yield state output distributions is called DNN-HMM. This is also known as hybrid system in the literature.

Figure 2.2 illustrates the structure of deep neural network (DNN) [199].

Input features consist of observations from several consecutive frames. These are fed into a sequence of hidden layers, where each hidden layer applies a linear transform, followed by an element-wise non-linear transform, such as sigmoid function. Softmax function is used in the output layer to yield a valid probability. Tied triphone states derived from decision tree are used as target.

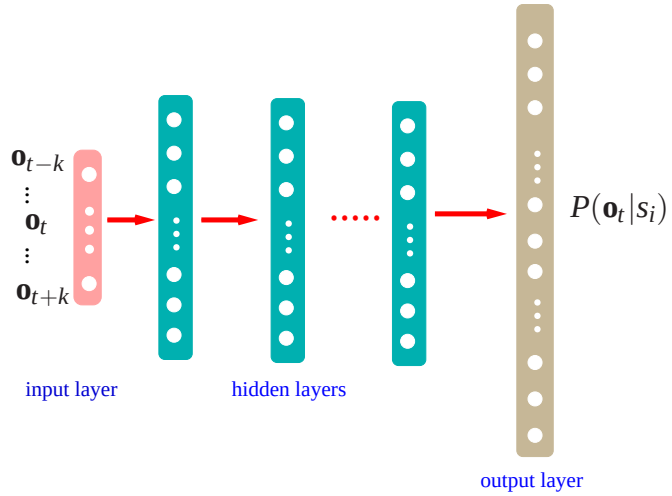


Fig. 2.2 *An illustration of deep neural network with multiple hidden layers. The input is acoustic feature and output is the tied states.*

Training of DNNs consists of two stages, pretraining and fine-tuning. Pretraining aims to find a good initialisation for the following fine-tuning stage. There are two types of pretraining used for acoustic model training. The first type is based on Restricted Boltzman Machine [96]. The other type is based on discriminative layer-wised pretraining. The later method was found to converge faster in the fine-tuning stage [198]. In the fine-tuning stage, cross entropy is used as objective function and standard error backpropagation algorithm is adopted based on stochastic gradient descent.

The state posterior probability $P(s_i|\mathbf{o}_t)$ is calculated from DNN. However, in speech recognition, the observation likelihood $p(\mathbf{o}_t|s_i)$is required, which can be obtained by,

$$p(\mathbf{o}_t|s_i) = \frac{P(s_i|\mathbf{o}_t)p(\mathbf{o}_t)}{P(s_i)} \tag{2.16}$$

where $p(\mathbf{o}_t)$ can be ignored during decoding. $P(s_i)$ is the prior probability of state s_i.

There have been a lot of efforts to improve performance of the DNN-HMM systems in recent years. Lattice based sequence training using MMI or MBR criteria were investigated in [115, 214, 232], second order (Hessian free) optimisation was examined in [116]. In addition, filter bank feature were found to outperform features like MFCC [161] for acoustic modelling. Recently, recurrent neural networks were found to yield gains over DNNs [83, 191, 162]. Deep neural network and recurrent neural network can also be used for language modelling, which will be introduced in Chapter 3.

2.2.3 Tandem System

Instead of using the output of neural network to calculate likelihood directly, there is another branch in acoustic modelling that makes use of neural network: Tandem system [95, 253].

The neural network is used to extract acoustic features, which are concatenated with standard acoustic features to form "new" acoustic features for subsequent GMM-HMM training. Figure 2.3 shows an example of Tandem system. The neural network is trained based on input feature vectors consisting of several consecutive frames and output targets being context independent (CI) phones or context dependent (CD) states. This is the same as the neural network in a Hybrid system, except that there is a bottleneck layer for extracting features. The acoustic features used in Tandem systems consist of two parts. One are traditional acoustic features such as MFCC, PLP. The other are bottleneck features extracted from neural network.

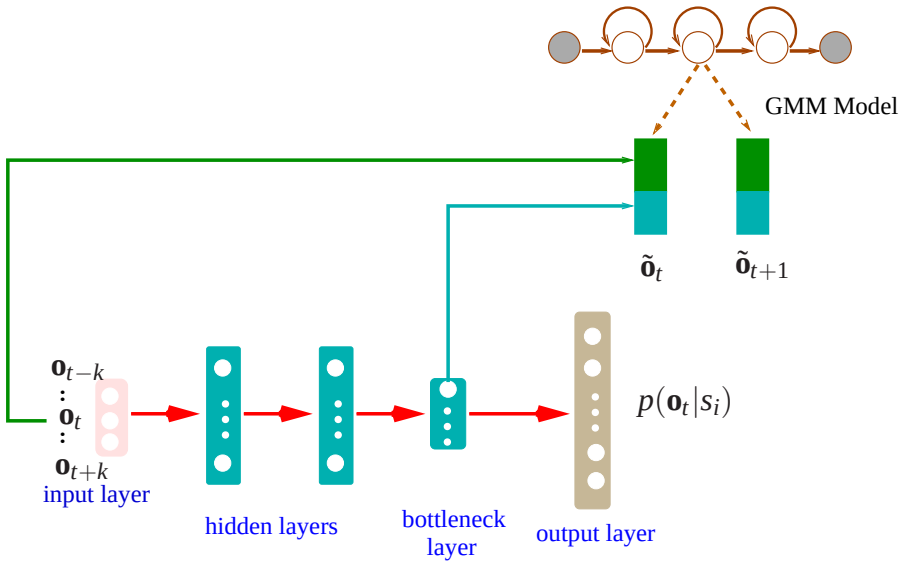


Fig. 2.3 *An illustration of deep neural network. Neural network is used to extract bottleneck feature from the output of hidden layer and this bottleneck feature will be concatenated with the standard acoustic feature to form Tandem feature and fed into GMM-HMM systems*

In [95], the Tandem system is first introduced into speech recognition. The posterior of phone is modelled and combined with the traditional acoustic features to form high dimensional Tandem features. These features are also called bottleneck features in the literatures. In [86], bottleneck features from the output of a hidden layer (before the non-linear function in the hidden nodes) were used to substitute posterior features and it yielded performance gain. The dimensionality of bottleneck features, typically 26, is relatively low compared with the number of nodes in the output or other hidden layers. The GMM-HMM system trained using this kind of feature is called a Tandem system in the literatures. Traditional techniques such as MLE and discriminative training can be applied to Tandem systems without modification.

2.2.4 Joint Decoding

In many systems, various acoustic models are trained separately and then combined for better performance. Joint decoding [233] and Confusion network combination (CNC) [63] are two popular methods to combine systems together for improved performance. Joint decoding combines systems during decoding. Taking Tandem and Hybrid system combination as an example, given a speech frame $\mathbf{o}_t$, the log-likelihood of state s_i in joint decoding can be computed as,

$$L_J(\mathbf{o}_t|s_i) \propto Z(s_i) + \lambda_H L_H(\mathbf{o}_t|s_i) + \lambda_T L_T(\mathbf{o}_t|s_i) \tag{2.17}$$

where $Z(s_i)$ is normalisation term and normally set to 0, $L_T(\mathbf{o}_t|s_i)$ and $L_H(\mathbf{o}_t|s_i)$ are the log-likelihood from Tandem and Hybrid systems for state s_i given observation $\mathbf{o}_t$.

The joint decoding system used in this chapter is shown in Figure 2.4.

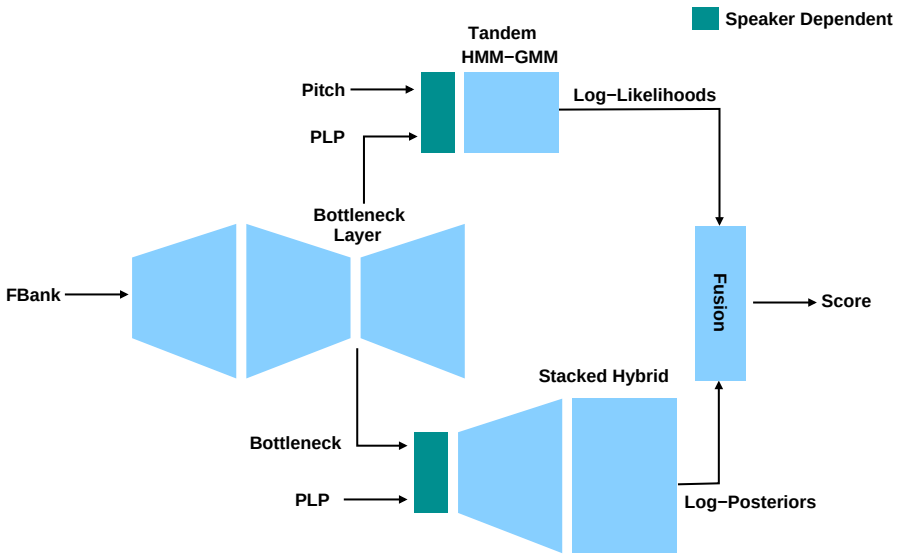


Fig. 2.4 *An illustration of joint decoding system for acoustic modelling. A Tandem and Hybrid systems can be combined on score level.*

λ_T and λ_H denote weights for Tandem and Hybrid systems. In this chapter, the following empirical values were used for joint decoding.

$$\begin{aligned} \lambda_T &= 0.25 \\ \lambda_H &= 1.0 \end{aligned} \tag{2.18}$$

In confusion network decoding [145, 63], confusion networks are obtained by decoding Hybrid and Tandem systems first. The confusion networks from two or more systems are aligned and combined, and the hypothesis word with highest posterior probability is chosen for output. These two methods normally give comparable performance [233]. However, joint decoding saves the compute time as it only requires a single decoding. Furthermore,

compact lattices can be generated by joint decoding, which can be used for lattice rescoring with better models to further improve performance.

2.2.5 Acoustic Model Adaptation

For statistical pattern classification, it is important that the training data is representative of the testing data. Otherwise, a serious performance degradation may be caused due to the mismatch. In speech recognition, test data always contains unexpected factors such as new speakers. In order to make acoustic model more suitable for new speakers, speaker adaptation is applied widely to transform the speaker independent acoustic models into speaker adapted models. It helps to reduce the mismatch between the model and the test data. It is worth noting that in this section, only speaker adaptation is discussed. However, these methods can aslo be applied to adaptation with other factors such as gender, environments, and so on.

Speaker adaptation can be divided into supervised and unsupervised adaptation according to whether references for adaptation data are available or not. In the supervised adaptation, the correct transcription is given and used for adaptation. However, in most applications, it is very costly or impractical to obtain references for adaptation data. Unsupervised adaptation is adopted instead where these hypotheses are generated by an ASR system. In this thesis, the unsupervised adaptation is considered because it is more practical and useful in real life.

A number of adaptation algorithms have been proposed for GMM-HMM system with consistent and significant performance gains. There are three broad categories of adaptation methods [240], maximum a posterior (MAP) [74] which estimates the model parameters using Bayesian inference, linear transform techniques such as maximum likelihood linear regression (MLLR) [133], and techniques using subspace to represent speaker model, such as cluster adaptive training (CAT) [70] and eigen voice adaptation [126].

Speaker Adaptation on DNN-HMM System

There is also a strong interest in speaker adaptation for DNN-HMM systems to improve performance. One of the earliest works on adaptating DNN-HMM systems in ASR can be traced back to the 1990s [166]. Along with the revived interests in Hybrid system, the adaptation of DNNs has been studied intensively during the past several years. Several methods for speaker adaptation with DNN-HMMs are reviewed here.

One method is to transform input features. These features transformed by CMLLR from GMM-HMM systems were used for normalisation of different speakers in [198]. In [234], FE-CMLLR [135] added on Tandem system was investigated. In FE-CMLLR, the linear transform is determined by the posterior from GMM, rather than regression tree when multiple transforms are used. The transformed feature will be used as input for the training of DNN to get a speaker adapted DNN-HMM system.

Another method is to append a speaker related feature as an auxiliary feature, such as i-vector feature for speaker recognition [193, 113] and speaker code [2], to standard acoustic

features. The acoustic adaptation using i-vector is illustrated in Figure 2.5. The blue block in the input layer represents the speaker related i-vector.

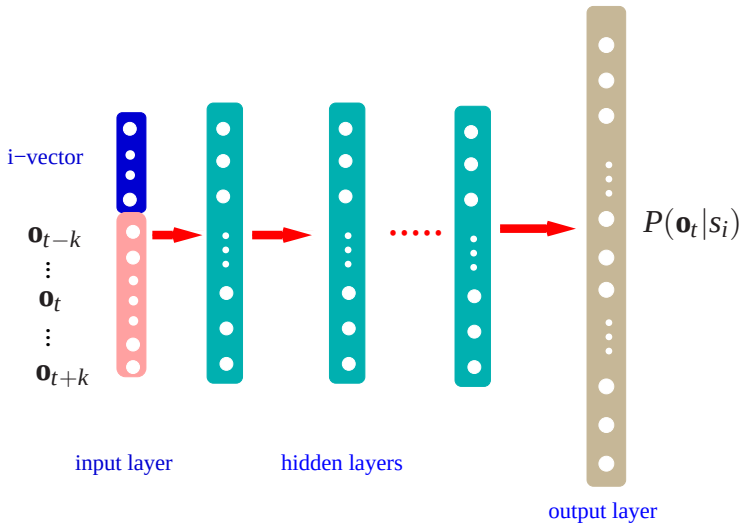


Fig. 2.5 *An illustration of deep neural network based acoustic model adaptation using i-vector. The i-vector feature is used as appended feature in the input layer of deep neural network for speaker adaptation.*

The i-vector is estimated on the training data for each speaker first. During training, the i-vector is appended to the standard acoustic features for each speaker. At test time, for each speaker, its i-vector is estimated, and appended with the acoustic features. The speaker code approach adopts a similar structure, but the speaker representation vector is derived from the DNN. This structure is also adopted for the neural network language model adaptation and more details can be found in Chapter 6.

The DNN model can also be adapted directly [198, 247, 254]. In [198], feature-space Discriminative Linear Regression (fDLR) was applied, where an additional adaptation layer is added directly after the input layer. The adaptation layer linearly transforms input feature, which is a CMLLR-like transform. Because of adaptation data sparsity, the transform is shared between different frames. During adaptation, only the adaptation layer is updated. In case of very limited adaptation data, diagonal transform matrix can be applied [198]. As an improvement, [134] adds L2 regularisation using weight decay and updates the input, output layer or all layers. In [254], a KL-divergence based regularised adaptation was proposed. In [225, 243], a cluster adaptive adaptation was investigated. In [222], the output of activation is used for speaker adaptation.

2.3 Language Model

Language modelling aims to compute the probability of any given word sequence $P(\mathcal{W})$. Language models play an important role in many applications including speech recogni-

tion, machine translation and spoken language understanding. For speech recognition, they impose constraints on the possible word sequence by computing the probabilities of the sequence.

The probability of a word sequence $\mathcal{W} = \{w_0, w_1, w_2, ...w_N\}$ can be decomposed into cascading probabilities using the chain rule. The overall probability can be written as products of conditional probabilities for each word given its history.

$$\begin{aligned} P(\mathcal{W}) &= P(w_0, w_1, w_2, ...w_N) \\ &= \prod_{i=1}^{N} P(w_i | w_{i-1}, ...w_1, w_0) \end{aligned} \tag{2.19}$$

where N is the valid length of word sequence $\mathcal{W}$, w_0 is always the symbol of sentence start, e.g. <s> and w_N is always the sentence end symbol, e.g. </s>. Language models are trained on a set of training corpus to estimate the probability of $P(w_i|w_{i-1}, ...w_1, w_0)$. However, for any applications with even a moderate vocabulary size, the number of parameters for this model is prohibitively large and impractical to store and compute all combinations of word w_i and history $(w_{i-1}, ...w_1, w_0)$. Hence, the history $(w_{i-1}, ...w_1, w_0)$ is normally grouped to an equivalent class. Assuming Φ denotes the function used for clustering histories, the word probability can be rewritten as,

$$P(w_i|w_{i-1}, ...w_1, w_0) \approx P(w_i|\Phi(w_{i-1}, ...w_1, w_0)) \tag{2.20}$$

The class function $\Phi(\cdot)$ clusters the history based on some criterion and probabilities among the equivalent classes being shared. This reduces the number of parameters significantly and allows reliable parameter estimation; e.g. n-gram language model introduced in the following section clusters histories with the same previous $n-1$ as equivalent class.

The quality of a language model can be measured directly by perplexity. Given a word sequence $\mathcal{W}$ including N words, the PPL of language model can be calculated as,

$$PPL = 2^{-\frac{1}{N}\log_2(P(\mathcal{W}))} = 2^{-\frac{1}{N}\log_2(P(w_0, w_1, ..., w_N))} \tag{2.21}$$

Language model with a lower perplexity means it gives a more accurate prediction, with less uncertainty and confusion. The improvement in perplexity is expected to reflect in speech recognition by reducing word error rate, although this is not always true. Hence, word error rate provides an important metric to evaluate the quality of language model in speech recognition.

There are a range of language models in the literature. In this chapter, we mainly focus on n-gram LM, which is probably the most popular language model of the past several decades. It is a simple model with good generalisation on unseen data and efficient parallelisation algorithm on large amount of training data. Although good performance is achieved in various tasks, n-gram LM has two well-known issues: data sparsity and long term history. In order to handle these issues, several variants derived from n-gram LM are discussed in Section 2.3.4 and 2.3.5 An inherently different type of language model, neural network based language model, is introduced in Chapter 3 to address these two issues.

2.3.1 *n*-gram Language Model

As discussed above, classing function $\Phi(\cdot)$ is used to cluster similar histories and share parameters. In *n*-gram language model, the history is clustered according to the previous $n-1$ words, which is,

$$\Phi(w_0, w_1, ...w_{i-1}) = \Phi(w_{i-n+1}, ..., w_{i-1}) =< w_{i-n+1}, ..., w_{i-1} > \tag{2.22}$$

w_{i-n+1}^{i-1} is used to denote a word sequence $w_{i-n+1}, ..., w_{i-1}$ for simplicity, which is called the *n*-gram history. The *n*-gram language model probability can be written as

$$P(w_i|w_0^{i-1}) \approx P(w_i|w_{i-n+1}^{i-1}) \tag{2.23}$$

In practice, at the begining of a sentence, the sentence start symbol <s> is inserted. In addition, a sentence end </s> is appended to the end of sentence. Hence, the probability of the first word is expressed as $P(w_1|$ <s> $)$, and an additional sentence end probability is estimated. *n* is normally called the order of *n*-gram language model. When *n* is equal to 1, unigram LM is constructed; and bigram LM is built by setting *n* equal to 2. Trigram (3-gram) and 4-gram LMs are used widely in speech recognition systems. Under this *n*-gram approximation, it is straightforward to estimate the probability of *n*-gram LM using maximum likelihood (ML) criterion as,

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^{i})}{C(w_{i-n+2}^{i})} \tag{2.24}$$

where $C(w_{i-n+1}^{i})$ is the frequency of the word sequence w_{i-n+1}^{i} in training data. Equation 2.24 gives a valid probability as,

$$C(w_{i-n+2}^{i}) = \sum_{w_{i-n+1}} C(w_{i-n+1}^{i}) \tag{2.25}$$

This simple estimation of *n*-gram LM probability has several issues. In order to robustly estimate the probabilities, sufficient coverage of possible word sequence $C(w_{i-n+1}^{i})$ is required. However, the *n*-gram LM suffers data sparsity problem even only considering the previous $n-1$ words. Taking trigram (i.e. 3-gram) LM for an example, for a moderate vocabulary with 20K words, there are as many as $20000^3 - 1 = 8e^{12} - 1$ free parameters to estimate. Furthermore, many words triplets occurring in test time don't appear in the training corpus. This results in a zero probability according to Equation 2.24. Various smoothing techniques have been developed for robust parameter estimation, which are introduced in the next section. The second issue lies in the n^{th} order Markov assumption in Equation 2.22. The predicted word probability is only dependent on the preceding $n-1$ words, while the longer range context is ignored. The long term history may contain useful information for prediction. There are plenty of works to mitigate this issue, which are briefly reviewed in Section 2.3.5.

2.3.2 Smoothing

Smoothing is applied in the *n*-gram LM to adjust the probability more robust, and avoid zero probability during estimation. The essential idea of smoothing is taking out some probability mass from the frequently seen *n*-grams and redistributing the mass to the infrequently seen or unseen *n*-grams. There are various methods to smooth the *n*-gram LM probability. Several popular methods are briefly reviewed here. A more comprehensive study on smoothing for language model is referred to [35, 255].

- **Katz Smoothing** Katz smoothing [114] extends the idea of the Good Turing estimate [77] in language modelling. The Good Turing states that an *n*-gram count that occurs r times, should be treated as if it occurs r^* times,

$$r^* = (r+1)\frac{n_{r+1}}{n_r} \tag{2.26}$$

  where n_{r+1} is the number of *n*-gram appears $r+1$ times in the training data. When this idea is applied in language modelling, Katz smoothing is applied as below: if an *n*-gram has been seen in the training data, the probability of the predicted word is discounted by multiplying a ratio; otherwise, the probability of the predicted word is calculated with a lower order of *n*-gram probability, where the most distant word is discarded. This scheme is also called back-off. The lower order Katz probability distributions are called back-off distributions. The back-off scheme in a trigram LM is illustrated in Figure 2.6.

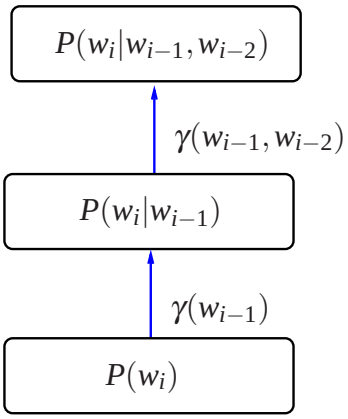


Fig. 2.6 *An illustration of back-off scheme in a trigram LM. When the trigram LM probability is not existed, it backs off to bi-gram LM probability. Uni-gram LM probability will be used if the bi-gram LM probability is not existed as well.*

A general form of back-off scheme can be written as,

$$P(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \alpha(w|w_{i-n+1}^{i-1}) & if \quad C(w_{i-n+1}^{w_i}) > 0 \\ \gamma(w_{i-n+1}^{i-1})\beta(w_i|w_{i-n+2}^{i-1}) & else \end{cases} \tag{2.27}$$

where $\alpha(w_i|w_{i-n+1}^{i-1})$ is the discounted n-gram probability, $\beta(w_i|w_{i-n+2}^{i-1})$ is the back-off probability from lower order of $n-1$-gram LM. $\gamma(w_{i-n+1}^{i-1})$ is the normalisation term to guarantee $P(w_i|w_{i-n+1}^{i-1})$ be a valid probability.

$$\gamma(w_{i-n+1}^{i-1}) = \frac{1 - \sum\limits_{w_i:C(w_{i-n+1}^{i})>0} \alpha(w_i|w_{i-n+1}^{i-1})}{\sum\limits_{w_i:C(w_{i-n+1}^{w_i})=0} \beta(w|w_{i-n+1}^{i-1})} \tag{2.28}$$

Various smoothing techniques discussed next mainly differ in the choice of discounted probability $\alpha(w|w_{i-n+1}^{i-1})$ (e.g. absolute discounting) and back-off probability $\beta(w_i|w_{i-n+2}^{i-1})$ (e.g. Kneser-Ney smoothing).

- **Absolute Discounting** Similar to Katz smoothing, absolute discounting [167] also computes the probability using back-off scheme. The main difference lies in the discounting way. Instead of multiplying a ratio, a fixed discount value $C \leq 1$ is subtracted from the n-gram count. This discount constant C can be calculated using leave-one-out [167], where,

$$C = \frac{n_1}{n_1 + 2n_2} \tag{2.29}$$

  and n_1 and n_2 are the frequencies of n-grams appears one and two times respectively. There are different ways to select the discount value C [35].

- **Kneser-Ney Smoothing** Kneser-Ney (KN) smoothing [120] is similar to absolute discounting in that it also subtracts a discount value $C \leq 1$ for nonzero n grams. The difference lies in the lower back-off order probability. Kneser-Ney smoothing adopts a smoothed ML estimation for lower order n-gram based on an important observation. The back-off should be optimised for lower count or unseen case. Taking "San Francisco" as an example, The word "Francisco" may have a very high probability in terms of unigram. However, it always occurs after "San". Hence, for a bigram LM, it should back off to a low unigram probability when it follows other words except "San", where the bigram probability is used directly; even "Francisco" has a high frequency in the training corpus. [78] presents a modified version of KN smoothing by introducing different discount value C for different n-gram counts. This modified KN smoothing is reported to be the best smoothing technique for word based language model [78].

The estimation of n-gram LM mainly involves the collection of n-gram counts and parameter smoothing. This is suitable for parallel computation. Hence, a large amount of data can be used for training. There are many engineering works carried out for efficient n-gram

LM training on large amounts of data [213, 25, 92, 91]. The advantages of easy implementation, fast training and good generalisation on unseen data make *n*-gram language models the most popular and dominant language model over the last several decades.

2.3.3 Language Model Interpolation

Language model interpolation is widely used for combination of multiple language models. Individual language model is trained on corpus from different domains. These language models can be combined in test time. Linear interpolation and log linear interpolation [118, 87] are two common interpolation methods. Linear interpolation is shown as,

$$P(w|h) = \sum_{k=1}^{K} \lambda_k P_k(w|h) \tag{2.30}$$

where K is the number of language model component and λ_k is the interpolation weight for the kth language model $P_k(w|h)$. The parameter λ_k can be optimised via EM algorithm on a held-out set [57].

Assuming that the held-out set consists of N words and there are K language models for linear interpolation; Then the objective function is,

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log(\sum_{k=1}^{K} \lambda_k P_k(w_i|h_i)) \tag{2.31}$$

where $\boldsymbol{\theta}$ is the set of interpolation weights λ_k to be optimised. This formula can be viewed as a mixture model and each mixture component is a separate language model. Based on the previous discussion of the EM algorithm, the auxiliary function can be written as,

$$\mathcal{Q}(\boldsymbol{\theta}; \boldsymbol{\theta}^{(t)}) = \sum_{i=1}^{N} q(\lambda_k^{(t)}) \sum_{k=1}^{K} \log(\lambda_k P_k(w_i|h_i)) \tag{2.32}$$

where $q(\lambda_k^{(t)})$ is the posterior probability of the kth language model component given all training words given the interpolation at the tth iteration, It can be computed as,

$$q(\lambda_k^{(t)}) = \frac{\lambda_k^{(t)} \sum_{i=1}^{N} \log(P_k(w_i|h_i)}{\sum_{j=1}^{K} \lambda_j^{(t)} \sum_{i=1}^{N} \log(P_j(w_i|h_i))} \tag{2.33}$$

By maximising Equation 2.32 in terms of interpolation weights λ_k, the update of interpolation weights in the $t+1$th iteration can be obtained as,

$$\lambda_k^{(t+1)} = \frac{q(\lambda_k^{(t)})}{\sum_{j=1}^{K} q(\lambda_j^{(t)})} \tag{2.34}$$

The above update formula can be used for iterative update until the objective function in Equation 2.31 converges.

Log linear interpolation can be written as,

$$P(w|h) = \frac{1}{Z(h)} \prod_{k=1}^{K} P_k(w|h)^{\lambda_k} \tag{2.35}$$

where $Z(h)$ is the normalisation term and λ_k is the interpolation weight to estimate. The normalisation term $Z(h)$ can be calculated by summing over the whole vocabulary $\mathcal{V}$,

$$Z(h) = \sum_{w \in \mathcal{V}} \prod_{k=1}^{K} P_k(w|h)^{\lambda_k} \tag{2.36}$$

This is a typical log-linear model and generalised iterative scaling [52] can be used for optimisation.

Language model interpolation can be naturally extended for language model adaptation [11]. The interpolation weights of language component are optimized on the in-domain data for adaptation purpose. The advantage is that there are only few parameters to be estimated. These parameters can be estimated robustly given several hundreds of words.

2.3.4 Improved *n*-gram Language Model

Despite the sophisticated smoothing technique in *n*-gram LM, data sparsity is always an issue with the increasing of *n*-gram order. Many attempts have been explored to mitigate the issue, either by reducing the number of parameters via different clustering or introducing richer contextual information. Several typical methods are briefly review in this chapter.

- **Class based language model** The class based language model [23, 147] aims to mitigate the data sparsity problem by sharing data. In the word based *n*-gram LM introduced above, each individual word is viewed different and treated alone. However, there are many words sharing the same context due to their inherent similarity, such as Monday and Tuesday, son and daughter. These words with contextual or syntactic similarities are clustered to one class. The *n*-gram LM probability is estimated based on class instead of word. The number of classes is expected to be largely smaller than vocabulary size, so as to obtain a more robust parameter estimation. For a class based trigram language model, the predicted probability can be written as Equation 2.37,

  $$P(w_i|w_{i-1}, w_{i-2}) = P(w_i|c_i)P(c_i|c_{i-1}, c_{i-2})P(c_{i-1}, c_{i-2}|w_{i-1}, w_{i-2}) \tag{2.37}$$

  where c_i is the class assignment of word w_i. Normally, each word is assigned to a single class and one class contains several words. Hence, the third term $P(c_{i-1}, c_{i-2}|w_{i-1}, w_{i-2})$ can be viewed as deterministic and always 1. There are many ways to estimate the assignment from word to class, either from linguistic knowledge such as POS tag [169], or purely data driven [23, 119, 175, 12, 147, 246].

The class based *n*-gram LM is normally interpolated with word based *n*-gram LM to get the best performance. The class based *n*-gram LM works well on small amounts of data. However, the improvement disappears with the increase of training data [78].

- **Random forest based language model** The random forest based language model [245] is derived from decision tree based language model [177]. The histories w_{i-n+1}^{i-1} are classified to many equivalent classes by decision tree by asking questions. These questions can be very general, such as whether the most distant word is some specific word. Histories in the same class share the distribution over the predicted probability. An improved decision tree algorithm was proposed in [245] by training a complete decision tree on the training data and pruning it on the held-out data with the KN smoothing technique. Several decision trees are constructed by introducing some randomisation. The probabilities from different decision trees are aggregated. The experiments in [245] show that a 10% PPL reduction and 6% WER reduction is obtained. The training of random forest language model on larger data obtained a 3% WER improvement [215] .

  Random forest based language model has a large potential to work well if the good partition of history can be found by asking general questions. However, the drawback is the heavy computation during training, e.g. in [245], and 100 decision trees were built. Besides, the improvements seem to decrease when there is more training data.

- **Structured language model** The structured language model [30, 29] aims to explore the syntactic structure in natural language to improve language modelling. It is based on the assumption that syntactic structure can filter out irrelevant words and the head words provide better prediction for the next word. The advantage of structured language model also comes from long term information instead of the last $n-1$ words. A partial parse is adopted on the history to find the syntactic order of a sentence.

  The structured language model is reported to give large perplexity reduction (11% in [30] and up to 24% in [29]). However, it is questionable when it is applied to more spontaneous spoken language application as it is difficult to discover proper syntactic structure. It is also difficult to apply to speech recognition systems when the recognised result contains many errors. Besides, the improvement diminishes when linearly interpolating with other long term language models such as the cache based language model [78] introduced in Section 2.3.5.

- **Factored language model** The factored language model was first introduced in 2003 [15]. In the factored language model, each word is represented as a feature vector. Generalised back-off is used for smoothing in FLM. This language model is very convenient to incorporate informative features such as root, stem, part-of-speech and morph. This is quite useful when there is a scarcity of in-domain training data or morphological information. It is applied widely for language models such as Arabic. [231] reported 2% absolute WER improvement, and 0.6% absolute improvement was reported in [212].

2.3.5 Beyond *n*-gram Language Model

In this section, the efforts to overcome the other drawback of *n*-gram LM, i.e. the drop of long term history, are introduced. Two representative language models are reviewed, which are cache based and maximum entropy based language models.

- **Cache based language model** The cache based language model was introduced in 1990 [125]. It is based on the hypothesis that the words used recently have a higher probability to appear soon than either their overall frequencies or the predicted probabilities from standard *n*-gram LM. To capture this feature, the words appearing in the recent past are cached and used to estimate the probability for the cache component. This work was originally carried out on the part-of-speech (POS) based language model. Each POS maintains a cache with room for 200 words. Each word is assigned to its POS tag and then stored in the corresponding cache. When there are more than 5 words classified to the same POS tag, the cache model is activated and used to estimate the cached probability. The probability from the cache is further interpolated with standard *n*-gram LM. The word probability consists of two parts: the first part is from *n*-gram LM and the other part is from the cache model. A significant perplexity reduction was reported in [125]. In [111], the cached based trigram LM (also known as dynamic model) was estimated based on recent history. The dynamic model is again interpolated with the static trigram word based LM. This dynamic model resulted in a 23% reduction in perplexity, and up to 24% WER reduction, after collecting the first several hundreds of words for the document.

- **Maximum entropy based language model** Maximum entropy based language model [130, 187] belongs to the family of the exponential language model [33]. The general form of the maximum entropy language model can be written as,

$$P(w|h) = \frac{1}{Z(h)} \exp[\sum_{k=1}^{K} \lambda_k f_k(w,h)] \tag{2.38}$$

where $f_k(w,h)$ is arbitrary feature defined by the word w and its history h, and λ_k is the set of parameters to estimate. $Z(h)$ is the normalisation term to ensure a valid probability.

$$Z(h) = \sum_{w \in \mathcal{V}} \exp[\sum_{k=1}^{K} \lambda_k f_k(w,h)] \tag{2.39}$$

It is worth noting that the log linear interpolation discussed in Equation 2.35 is a special case of the maximum entropy language model, where the feature $f_k(w,h)$ is chosen as the log probability of each language component $\log(P_k(w|h))$.

A unique ME solution is guaranteed to exist for this problem. Generalised iterative scaling [52] can be applied to get the global optimal.

Maximum entropy introduces the trigger pair for feature design. The trigger pair is defined on two highly correlated sequences A and B. $A \rightarrow B$ denotes a trigger pair,

where A is the trigger and B is the triggered sequence. When A occurs in the history, it triggers B by affecting its probability. There are a large number of possible trigger pairs according to this definition. The trigger pair can be filtered out by considering their co-occurrence frequency. One good measure can be their average mutual information [130]. An elegant feature of maximum entropy language model is that when the n-gram counts are used as trigger pairs, the unique ME solution is the maximum likelihood solution. It was found that the self trigger pairs ($A = B$) are very informative [187]. The long term information is captured by using trigger pairs. The maximum entropy based language model was reported to give 32-39 % PPL reduction and 10-14% WER reduction in [130, 187]. One drawback of this model is the computational complexity. Despite the effort on the speedup of training [79], it is still time-consuming and hard to use a large amount of training data. A regularised class based maximum entropy based model, also known as model M, was developed in 2009 [34]. This model gives state-of-the-art performance on broadcast new tasks [36].

- **Sparse non-negative matrix language model** More recently, a sparse non-negative matrix language model using skip-grams was proposed by Google [202, 203]. It is similar to the maximum entropy based language model as they are both able to model long term information by applying constraints on various features. However, in the non-negative matrix language model, the skip-grams feature is used. Additionally, rather than using log linear function, the non-negative matrix is used as follows,

$$\mathbf{y} = \mathbf{M}\mathbf{f} \tag{2.40}$$

where $\mathbf{f}$ defines a vector of skip-gram features, and $\mathbf{M}$ is a non-negative matrix including the parameter to be optimised. The output $\mathbf{y}$ is normalised to obtain a valid probability for the kth word using,

$$P(k|\mathbf{f}) = \frac{y_k}{\sum_{j=1}^{|\mathcal{V}|} y_j} \tag{2.41}$$

A significant improvement on perplexity compared to n-gram LM was obtained in [203], resulting in a comparative performance with recurrent neural network language. However, this model has not been reported on speech recognition yet, although this model is able to convert to the standard ARPA back-off based language model [174].

In this chapter, only certain typical language models from the last two decades are briefly reviewed. For a more comprehensive and thorough review, readers are referred to [188, 78].

2.3.6 Language Model Adaptation

The mismatch between training and test corpus is always an issue. The mismatch might be introduced from several aspects for language models. First, people have different customs

for choosing words and expression. Second, a range of speaking styles exist between training and test data, e.g. written English and spoken English are expected to be very different. Last but not the least, different domains have an underlying different vocabulary, even for the same domain, and the choice of language may evolve with time.

Given the inherent variability, the language model can be adapted to mitigate the effect of mismatch. Similar to acoustic model adaptation, language model adaptation can be divided into supervised and unsupervised adaptation. When the correct transcription is given, supervised adaptation is applied. However, in many applications, such as speech recognition, the reference is not available. The adapted text comes from the recognised hypothesis. In this way, unsupervised adaptation is carried out. Four popular language model adaptation methods are briefly introduced in this chapter. More details about language model adaptation can be found in [11]. Many of these techniques can be viewed as a natural extension of improved language model introduced in Chapters 2.3.4 and 2.3.5.

- **Interpolation** language model interpolation is widely used for language model adaptation. An individual language model is trained on corpus from different domains. These language models are combined in test time with linear interpolation as shown in Equation 2.30. The advantage of interpolation is that there are only a few parameters to be estimated. These parameters can be estimated robustly given several hundreds of words.

- **Context dependent interpolation** Global and fixed interpolation weights λ_k are used in the above section. [138, 141] studied the context dependent interpolation for adaptation, which is written as,

$$P(w|h) = \sum_{k=1}^{K} \lambda_k(h) P_k(w|h) \tag{2.42}$$

  The interpolation weight $\lambda_k(h)$ is related to its history h. It is impractical to estimate an interpolation weight for every possible history. In order to robustly estimate the interpolation weight, MAP weight adaptation and class context dependent weights were investigated in [138]. The former method uses context independent interpolation weight as prior and estimates the parameters based on training data under Bayesian framework. The latter reduces the number of parameters by clustering similar histories and sharing parameters.

- **Cache based LM adaptation** [121] extended the idea of cache based language model for the purpose of adaptation. A cache based language model is estimated based on the previous recognised words and then interpolated with a background language model. This cache based language model is able to capture the change of text by using the recent words, and good improvement was reported.

- **Topic based LM adaptation** Topic based language model was proposed in [76] and its expression is as below,

$$P(w|h) = \sum_t P(w|t)P(t|h) \tag{2.43}$$

where t is a latent topic variable which may be assigned to different topics, $P(w|t)$ is word probability given the topic, and $P(t|h)$ is topic posterior probability given history h. This topic-related model is able to predict word probability based on a long term history. However, it fails to make good use of the short term history, which is expected to attribute more to the prediction of the next word. Hence, the probability in Equation 2.43 is normally combined with the static n-gram language model. The maximum entropy model can be applied to make the estimated probability satisfy constraints from both n-gram LM and topic based language model. Generalised iterative scaling [52] is applied for optimisation. Various topic models can be used, including latent semantic analysis (LSA) [10], probabilistic latent semantic analysis (pLSA) [165] and latent Dirichlet allocation (LDA) [223].

- **Maximum entropy based LM adaptation** Maximum entropy based language model described in Chapter 2.3.5 can be viewed as an adaptive language model inherently [187]. The incorporation of triggers is able to capture the dynamic change in the use of words, and affects the predicted probability.

2.4 Search

Search aims to find the best recognition result according to Equation 1.1, by integrating various knowledge sources including the acoustic model, language model and lexicon. A decoder is implemented for search purpose and it is a crucial component in the speech recognition system. In academia, searching provides a straightforward way to validate the improvement of various techniques in speech recognition. In industry, searching is more crucial as the latency of search affects the custom experience directly. Hence, there is a strong desire to design a fast and robust decoder.

Two types of searching problems are discussed in this section. If the recognition result is obtained directly from the acoustic feature from scratch, the process is called first-pass decoding. A decoder can find the best hypothesis by incorporating acoustic and language models directly. However, first-pass decoding is computationally expensive due to the huge search space, especially when a large language model is applied. Lattice rescoring can mitigate the computational load by using multiple passes. The lattices are first generated with small models. The lattices define compact but sensitive search space. Sophisticated models are then applied to get more accurate results on lattices, which limits the search space and largely reduces computation.

2.4.1 Decoding

Given the acoustic model and language model, the decoder aims to search for the best path with the highest score as shown in Equation 1.1. The acoustic model based on HMM gives the probability of observations of a given phone (e.g. triphone) sequence. The lexicon specifies the phone sequence of each word. Language model calculates the probability of a word sequence. Moreover, one word may have multiple pronunciations. For the same phone sequence, the state sequences are different due to various state segmentations. By reviewing the Equation 1.2,

$$\begin{aligned}\hat{\mathcal{W}} &= \arg\max_{\mathcal{W}} p(\mathbf{O}|\mathcal{W})P(\mathcal{W}) \\ &= \arg\max_{\mathcal{W}} \Big(\sum_{S} p(\mathbf{O},S|Q)\sum_{Q} P(Q|\mathcal{W})P(\mathcal{W})\Big)\end{aligned} \tag{2.44}$$

where Q is the possible phone sequence given word sequence $\mathcal{W}$, and S is the possible state sequence given phone sequence Q, $P(\mathcal{W})$ gives the language model probability, $P(Q|\mathcal{W})$ is determined by the pronunciation probability and $p(\mathbf{O},S|Q)$ can be calculated from acoustic model.

It is computationally heavy and impractical to enumerate all possible word sequences considering the variable sentence length and the large vocabulary size. Approximations are introduced to handle this issue. The first approximation is to substitute the summation with maximisation as below,

$$\hat{\mathcal{W}} = \arg\max_{\mathcal{W}} \Big(\max_{S} p(\mathbf{O},S|Q)\max_{Q} P(Q|\mathcal{W})P(\mathcal{W})\Big) \tag{2.45}$$

Under this approximation, the most likely state sequence is kept and the corresponding word sequence is chosen as the recognised output.

An illustration of the decoding procedure is given in Figure 2.7. The recognised sentence can be broken down to smaller units, where each level of unit can be depicted by a specific model or constraint. A sentence is formed by a sequence of words depicted by the language model, each word consists of several phones, where the constraint is derived from lexicon. Each phone (e.g. triphone) has a few states, which corresponds to the HMM model. The state emission probability distribution can be modelled by GMM or DNN.

For implementation, a compact graph can be compiled to incorporate the information from acoustic model, lexicon and language model, which is called the decode network in the literature. The decode network can be built offline before test time. The aim of search (i.e. decoding) is to find the best possible path from the decode network. Width first (e.g. Viterbi) and depth first (A star) search algorithms can be applied [6]. Viterbi decoding is the most popular algorithm used for decoding. Given the whole observation sequence $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, ..., \mathbf{o}_T\}$, the partial best path (i.e. state sequence) at time t in state s_j is defined as,

$$\Psi_j(t) = \max_{\phi_0,..,\phi_{t-1}} p(\mathbf{o}_1, ..., \mathbf{o}_t, \phi_1, ..., \phi_t = s_j) \tag{2.46}$$

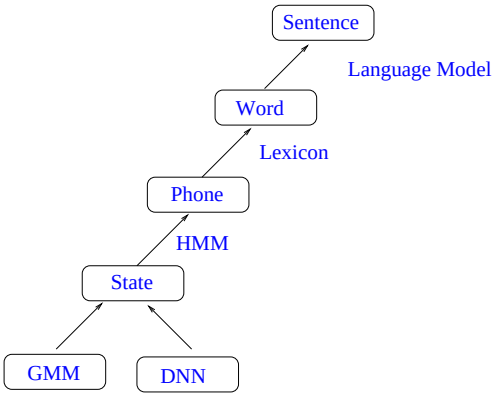


Fig. 2.7 *A decomposition of sentence in speech recognition. The sentence consists of word with language model; the word is a sequence of phone constrained by pronunciation lexicon, the phone is modelled by the state sequence in the HMM model. Each state can be modelled using GMM or DNN.*

The partial best path can be calculated recursively,

$$\Psi_j(t) = \max_i \{\Psi_i(t-1)a_{ij}\}b_j(\mathbf{o}_t) \tag{2.47}$$

The best state in time $t-1$ can be obtained via,

$$\phi_{t-1} = \arg\max_i \{\Psi_i(t-1)a_{ij}\} \tag{2.48}$$

with an initialisation

$$\Psi_1(0) = 1 \tag{2.49}$$

$$\Psi_j(1) = a_{0j}b_j(\mathbf{o}_1) \tag{2.50}$$

where state 0 is the entry state of HMM model. $S_j(T)$ gives the highest score for state s_j, ϕ_{t-1} computes the state of $t-1$ for the partial best path to time t at state s_j. The best path score in time T can be computed recursively using Equation 2.46 and its state sequence can be retrieved based on the recording from $s(t)$ with Equation 2.47.

Viterbi algorithm can be extended to continuous speech recognition, and an implementation of this algorithm is called token-passing algorithm [250, 168]. In this algorithm, each state maps to j in Viterbi algorithm. And each token contains a history path including the previous word sequence. Each state can contain multiple tokens with different histories to this state. The language model score is added between the transition from the end of one word to new word. Each jump consists of the state transition probability and acoustic likelihood from the state output probability. The path with highest likelihood is chosen as the

recognised result. The word sequence is retrieved by a backward search using Equation 2.47.

The complexity of decoding increases dramatically with the size of language model, especially the order of *n*-gram LM used. The search is time consuming. It is computationally unaffordable to use full search by keeping all the possible paths. Therefore, several approximations are introduced to speed up searching. For each state, only a fixed number of the highest tokens are kept. The other tokens with lower likelihood are discarded, although these discarded paths are possible to have a high likelihood with more observations. Beam search is used for pruning to reduce computation. A beam width can be specified and tokens with lower likelihood than the beam width are discarded. However, if the number of tokens is small or the beam width is too tight, the most likely path is probably pruned in an early stage, which damages performance although the decoding time is reduced significantly. Hence, the hyper parameters (beam width and number of tokens per node) control the tradeoff between decoding speed and accuracy.

In practical application, the log of probability is normally used for numerical stability. Due to the numerical ranges of the acoustic model and language model probabilities are quite different. The language model score is much smaller than acoustic model score. To overcome this issue, a language model scale is added in the log of language model probability, such as 14.0. A similar scale is applied for pronunciation probability as well. Besides, word penalty is introduced to control the number of words. Under these assumptions, the final recognised output in real speech recognition system is expressed as,

$$\hat{\mathcal{W}} = \arg\max_{\mathcal{W}} \Big(\log p(\mathbf{O}, S|Q) + \alpha \log P(Q|\mathcal{W}) + \beta \log P(\mathcal{W}) + \gamma L(\mathcal{W}) \Big) \tag{2.51}$$

where α is the pronunciation probability scale, β is the language model scale and γ is the word penalty. $L(\mathcal{W})$ is the length of the word sequence.

There are two types of decoders in the literature, which are dynamic and static decoders. The difference exists that whether the language model score is added during decoding or not. In a dynamic decoder, the decode network is compiled using acoustic model and lexicon, the language model score is added during decoding on the fly, while for the static decoder, the language model is compiled into decode network as well, which is often known as Weighted Finite State Transducer (WFST) based decoder. For example, HTK [251] adopted a dynamic decoder and Kaldi implemented a WFST based decoder [179]. There is also some work using the static decode network, while compiling the language model on the fly to reduce memory [100].

2.4.2 Lattice Rescoring

During decoding, the hypothesis with the highest likelihood is generated as output. A side product of decoding is lattice. Lattice is a compact graph containing possible paths during recognition. Figure 5.1 gives an example of lattice with a reference of "well I think that is true". Besides the correct path (labelled as a red line) in the lattice, there are a number of alternative paths with high likelihood. Each node in the lattice is associated with time, word,

acoustic likelihood and language model score information. In the lattice shown in Figure 5.1, the word information is moved to the edge for simplicity. In many applications, it is computationally difficult or impractical to use first-pass decoding for sophisticated acoustic and language models. Small models are used for first-pass decoding to generate the lattice. The lattice gives a large number of possible paths in a constrained search space. The more sophisticated and accurate models are applied in the constrained search space defined by the lattice and the improved recognition result can be obtained. Lattice is also very useful in many applications. For example, in MPE training, lattice is used to approximate all possible paths as shown in Equation 2.12. The lattice can also be used for the estimation of confidence score and key word spotting [237]. The confusion network decoding based on lattice can further reduce word error rate by finding the best path with lower word error.

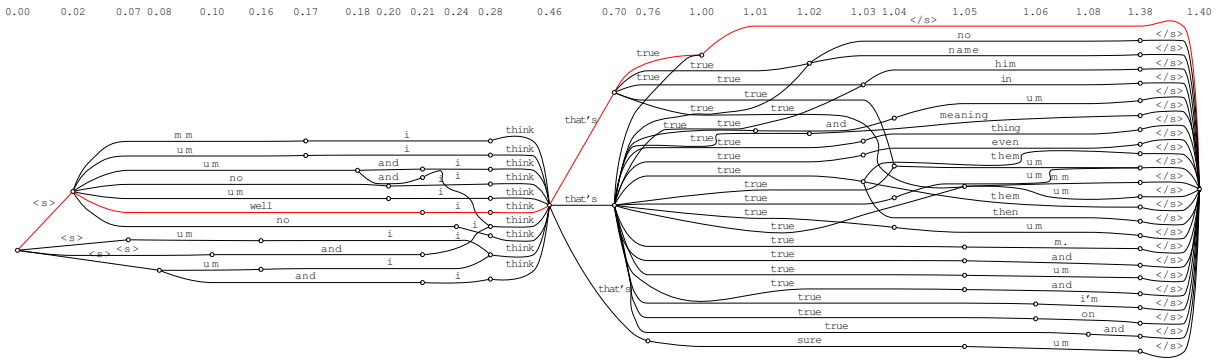


Fig. 2.8 *An example of lattice with reference "well I think that is true"*

2.4.3 Confusion Network Decoding

In Viterbi decoding, the hypothesis is obtained by optimising the objective function in Equation 1.2, which is rewritten as,

$$\hat{\mathcal{W}} = \arg\max_{\mathcal{W}} P(\mathcal{W}|\mathbf{O}) \tag{2.52}$$

The hypothesis $\hat{\mathcal{W}}$ is the most likely word sequence with minimum error rate at the sentence level. However, word error rate is normally used as the standard evaluation metric in speech recognition. Hence, there exists a mismatch between the objective function and evaluation metric. The output of Viterbi decoding is sub-optimal for word error rate.

In order to minimise the error at word level, Minimum Bayes's Risk on word level discussed before can be applied, and the word level posterior probability $P(w|\mathbf{O})$ is used. The word posterior probability can be calculated via the forward-backward algorithm in the lattice. The posterior probability of a link (i.e. an arc) in lattice can be expressed as,

$$P(l|\mathbf{O}) = \frac{\sum_{q\in Q_l} p(q,\mathbf{O})}{p(\mathbf{O})} = \frac{\sum_{q\in Q_l} p(q,\mathbf{O})}{\sum_{q\in Q} p(q,\mathbf{O})} \tag{2.53}$$

where Q_l is all paths through link l and Q is all paths in the lattice. Each link l contains information including the start and end time, word label and acoustic, pronunciation and LM score.

Each word may occur in multiple links in an overlapped time region in the lattice. Confusion network [145, 63] provides a feasible way to combine these posterior probabilities. First, the time dependent word posterior is computed as,

$$P(w|\mathbf{O},t) = \sum_{l_s<t<l_e, l_w=w} P(l|\mathbf{O}) \tag{2.54}$$

where l_s, l_e is the start and end time of link l and l_w is the word label of link l. In time t, the posteriors of links corresponding to word w are added. The word posterior $P(w|\mathbf{O},t)$ is clustered according to their time slots and phonetic similarities. A linear graph can be built by clustering the links. The phonetic similarity is also considered to form the confusion sets. Figure 2.9 gives an example of the confusion network converted from the lattice shown in Figure 5.1. The confusion network has a parallel structure in each time slot. The word with highest posterior probability in each time slot is chosen as output. Therefore, the recognised output in the confusion network shown in Figure 2.9 is "um well I think that is true um".

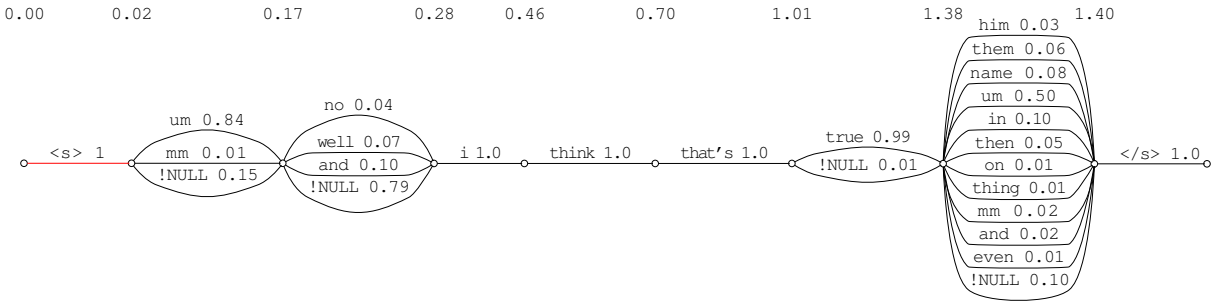


Fig. 2.9 *An example of confusion network for sentence with the reference "well I think that is true"*

Confusion network is very useful in practical application. It helps to further reduce word error by clustering links in lattice [63]. It can be also used for keyword spotting [256] and spoken language understanding [93].

2.5 Evaluation

The evaluation of speech recognition is important as it allows various models to be compared. The quality of speech recognition can be measured with word error rate (WER). WER is calculated by comparing the reference (correct sentence) and hypothesis. Two sentences can be aligned with dynamic programming for string alignment to minimise the Levenshtein distance. There are three types of errors, which are insertion, deletion and

subsection error. The calculation of WER can be expressed as,

$$WER = \frac{S+D+I}{N} * 100\% \tag{2.55}$$

where S is the number of substitution error, D is the number of deletion errors and I is the number of insertion errors. N is the number of words in the reference. A lower WER means a better speech recognition result. Figure 2.10 shows an example of WER computation. The reference and hypothesis word sequence are aligned. There are 7 words in total in the reference sentence. For the aligned sentences, there are 1 insertion error, 2 substitution errors and 1 deletion error. Hence the WER is $4/7 = 57.1\%$.

REF:	I HAVE NEVER		BEEN	AROUND	A	CAMPFIRE
HYP:	HAVE	NEVER	DREAM I	AM	A	CAMPFIRE
EVAL:	D		I	S	S	

Fig. 2.10 *An example of WER computation. There are 1 deletion error, 2 substitution errors and 1 insertion error when compared the reference "I HAVE NEVER BEEN AROUND A CAMPFIRE" and hypothesis "HAVE NEVER DREAM I AM A CAMPFIRE".*

2.6 Summary

This chapter reviews the fundamentals of speech recognition based on the Hidden Markov Model. First, the extraction of acoustic features, especially MFCC, PLP and FBank, is discussed. The preprocessing techniques for acoustic features are also described. The Hidden Markov Model (HMM) is then detailed, ranging from the model structure to parameter estimation. Two popular types of acoustic model, Gaussian mixture model (GMM) and deep neural network (DNN), are introduced. The adaptation for these two models is also briefly presented. Another important component in speech recognition, also the research topic in this thesis, the language model, is introduced. In this chapter, the discussion focuses on the most popular *n*-gram language model. There are two well-known issues for the standard *n*-gram language model, which are data sparsity and the *n*-gram assumption. Various smoothing techniques are introduced for robust parameter estimation, and extensions of standard *n*-gram LM are also presented to capture the long term information. The adaptation of *n*-gram language model is also discussed. When the acoustic model and language model are available, the speech recognition process turns to be a search problem, i.e. how to find the best hypothesis from the search space defined by the acoustic model, language model and pronunciation lexicon. Viterbi decoding is introduced as an efficient algorithm to find the best path. Confusion network decoding is also discussed to minimise word error rate. Finally, the evaluation of a speech recognition system is presented.

Chapter 3

Neural Network Language Models

Language models are crucial components in many speech and language processing applications including speech recognition. The aim of the language model is to estimate the probability of any given sentence as below,

$$\begin{aligned} P(\mathcal{W}) &= P(w_0, w_1, w_2, ..., w_N) \\ &= \prod_{i=1}^{N} P(w_i | w_{i-1}, ...w_1, w_0) \end{aligned} \tag{3.1}$$

where w_0 is the sentence start symbol <s> and w_N is sentence end symbol </s>. Due to their good performance and efficient implementation algorithm, *n*-gram LMs have been the dominant language modelling approach for several decades. However, there are two well-known issues associated with *n*-gram LMs [78]. The first is data sparsity. To address this problem, sophisticated smoothing techniques have been developed for robust parameter estimation as described in Chapter 2.3.2. The second issue lies in the n^{th} order Markov assumption. The predicted word probability is only dependent on the preceding $n-1$ words, and longer range context dependences are ignored. A range of work has been carried out to overcome these two issues. Many of them are the variants of the *n*-gram LMs as have been discussed in Chapters 2.3.2 and 2.3.5.

In this chapter, language models based on neural network are reviewed. Two widely used neural network language models (NNLMs), feedforward and recurrent neural network , are presented. Feedforward NNLMs [14] solve the data sparsity issue by projecting each word into a low-dimension and continuous space. RNNLMs [153] extend the concept to model long term history using a recurrent connection between input and hidden layers. Both of these NNLMs provide complementary information to standard *n*-gram LMs, and are often combined with *n*-gram LMs. They have become increasingly popular in recent years and promising results have been reported in a range of tasks and systems [196, 153, 154, 150, 216, 217, 61, 248].

3.1 Model Structure

There are various neural network structures that can be used for language modelling. In this section, three typical structures are reviewed, which are feedforward, recurrent and long short term memory neural network respectively.

3.1.1 Feedforward Neural Network Language Model

Feedforward neural networks (also known as multilayer perceptrons) were first introduced in the context of word-based language modelling by Bengio in 2003 [14]. They were further developed by Schwenk in terms of efficient training and application to speech recognition [196]. An *n*-gram feedforward NNLM can be constructed as shown in Figure 3.1 [1]. It is still an *n*-gram language model and the probability of any given sentence W can be written as,

$$\begin{aligned} P(\mathcal{W}) &= \prod_{i=1}^{N} P(w_i|w_{i-1},...w_1,w_0) \\ &\approx \prod_{i=1}^{N} P(w_i|w_{i-1},...w_{i-n+1}) \end{aligned} \tag{3.2}$$

The input consists of the previous $n-1$ words w_{i-n+1}^{i-1} and the target is the predicted word w_i. Each word is mapped to a single node in the input and output layer. 1-of-K coding (also known as one-hot representation) is used in the input layer, where only the node corresponding to the word is set to 1, the other nodes are 0. Rather than using the complete vocabulary, shortlists consisting of the most frequent words are normally used for input and output layers. The mapping between word and node in the input and output layers can be expressed as Equation 3.3. An out-of-vocabulary (OOV) input node can be used to represent any input word not in the input shortlist. Similarly, an out-of-shortlist (OOS) output node is added in the output layer to represent words not in the output shortlist. The use of shortlist can model the probabilities of OOV and OOS words in the input and output layers. A valid probability can be obtained via normalisation over the complete vocabulary. Besides, the shortlist in the output layer only contains the most frequent words, instead of the large complete vocabulary. The computation occurring in the output layer can then be reduced. The OOV and OOS nodes will be further discussed later in this section.

In the feedforward NNLM shown in Figure 3.1, each word is mapped to one node in both input and output layers using the following indexing,

$$\begin{aligned} \phi^{in}(w) &= l \qquad w \in \mathcal{V}^{in} \\ \phi^{out}(w) &= k \qquad w \in \mathcal{V}^{out} \end{aligned} \tag{3.3}$$

[1] In this chapter we only consider neural networks with a single hidden layer. Deep neural network based language models can be built in a similar way [4].

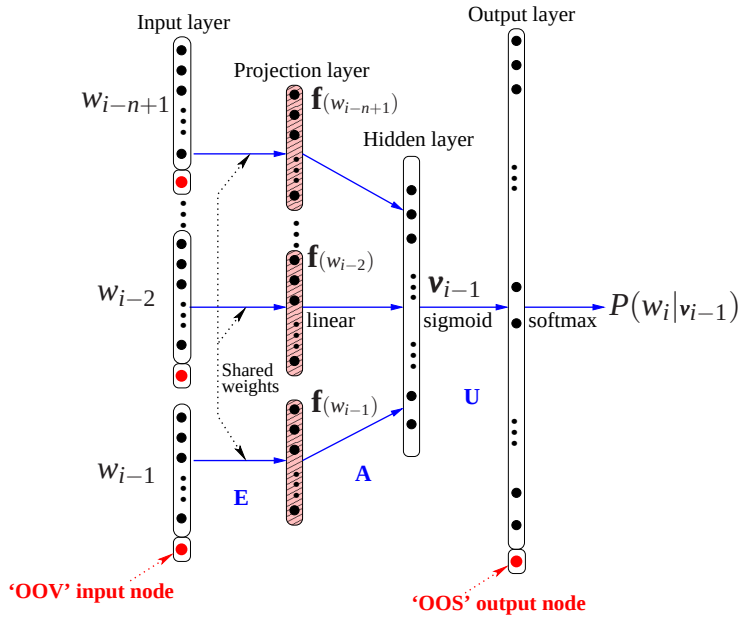


Fig. 3.1 *A 4-gram Feedforward neural network language model. The previous 3 words in the input layer are projected into a low-dimension vector with the shared projection layer. Feedforward neural network is applied as classifier to estimate the probability of current word* w_i

where l and k are the index of word w in the input and output layers. As stated before, the words not in the input word list are mapped to the OOV node, and words not in the output word list are mapped to the OOS node. The indexing functions for the OOV and OOS nodes can be defined as,

$$\begin{aligned} \phi^{in}(OOV) &= |\mathcal{V}^{in}| - 1 \\ \phi^{out}(OOS) &= |\mathcal{V}^{out}| - 1 \end{aligned} \tag{3.4}$$

where $|\mathcal{V}^{in}|$ and $|\mathcal{V}^{out}|$ are the size of input and output word lists respectively. The last nodes in the input and output layers are used to represent OOV and OOS words.

The indexing in the input and output layer may be different due to the difference of input vocabulary $\mathcal{V}^{in}$ and output vocabulary $\mathcal{V}^{out}$. 1-of-K coding is applied for each word w in the input layer to obtained the 1-of-K coding vector $\mathbf{r}^{(w)}$, whose j^{th} element is,

$$r_j^{(w)} = \begin{cases} 1 & j = \phi^{in}(w) \\ 0 & otherwise \end{cases} \tag{3.2.1}$$

Thus only one element in the 1-of-K coding vector $\mathbf{r}^{(w)}$ is 1 and all the others are 0. The 1-of-K coding vectors of the $n-1$ previous words are fed to the linear projection layer. The projection matrix $\mathbf{E}$ is shared over all input word vectors. Each word w in the input vocabulary $\mathcal{V}^{in}$ can be represented using a low-dimension vector $\mathbf{f}(w_i)$ by the projection

layer, whch is also known as word embedding in literature [157, 156].

$$\mathbf{f}(w) = \mathbf{E}^{\top}\mathbf{r}^{w} = \mathbf{e}^{\top}_{\phi^{in}(w)} \tag{3.5}$$

where $\mathbf{e}_i$ is the ith row vector of projection matrix $\mathbf{E}$ and $\phi^{in}(w)$ is the index of word w in the input layer.

The previous $n-1$ word embedding vectors are concatenated to form the vector $\mathbf{x}$, written as,

$$\mathbf{x} = \begin{bmatrix} \mathbf{f}(w_{i-n+1}) \\ \cdots \\ \mathbf{f}(w_{i-2}) \\ \mathbf{f}(w_{i-1}) \end{bmatrix}$$

$\mathbf{x}$ is fed to the hidden layer $\mathbf{A}$ and a non-linear function is applied as

$$\mathbf{v}_{i-1} = \sigma(\mathbf{A}^{\top}\mathbf{x} + \mathbf{b}_a) \tag{3.6}$$

where $\mathbf{b}_a$ is the bias vector in the hidden layer, and $\mathbf{v}_{i-1}$ is the output vector of the hidden layer. Sigmoid function is normally chosen as the non-linear function $\sigma(\cdot)$. The sigmoid function of the jth element can be written as,

$$\sigma(v_j) = \frac{1}{1+\exp(-v_j)} \tag{3.7}$$

The output of the hidden layer is multiplied with matrix $\mathbf{U}$ in the output layer,

$$\mathbf{z} = \mathbf{U}^{\top}\mathbf{v}_{i-1} + \mathbf{b}_o \tag{3.8}$$

where $\mathbf{U}$ is the matrix and $\mathbf{b}_o$ is the bias vector in the output layer. A softmax function is used at the output layer to get a positive and valid probability distribution over words in the output layer.

$$P(w_i|h_i) \approx P(w_i|w_{i-n+1}^{i-1}) = \frac{\exp(z_{\phi(w_i)})}{\sum_{w \in \mathcal{V}^{out}} \exp(z_{\phi(w)})} \tag{3.9}$$

The probability of the OOS node in the output layer needs to be handled specially to get a valid probability over the whole vocabulary. Usually, the probability mass of OOS words is re-distributed among all OOS words [172, 131].

One advantage of feedforward NNLMs is that they mitigate, to some extent, the data sparsity problem. The projection layer matrix $\mathbf{E}$ in the input layer is shared among history words. The number of model parameters increases linearly with a ratio of the projection layer size with the growth of the n-gram order, instead of increasing exponentially as in standard n-gram LM. The dimension of the word embedding vector and hidden layer size normally lies in the range between 100 and 500. A high order of n-gram feedforward NNLM can be constructed; for an example, 7 gram was used in [197]. Additionally, the standard n-gram LM treats each word as an atomic unit and any semantic connections are ignored. In

contrast, for neural language models, each word is represented as a vector in a continuous space and the distance between words is measured in the vector space. Words with similar context are expected to cluster together in vector space, so as to share parameters implicitly during training; e.g. "Saturday" and "Sunday" should be close in the vector space.

Based on the previous description, the impacts of input and output layer size on computation can be compared. In the input layer, the main computation is calculating the word embedding vectors for the previous $n-1$ words. Due to the use of 1-of-K coding, the computation doesn't increase when a larger input shortlist is applied. However, in the output layer, the computation increases significantly with the output layer size due to the matrix multiplication and softmax function shown in Equations 3.8 and 3.9. Hence, the input shortlist is often chosen to be the same as *n*-gram LM vocabulary and the output shortlist is normally a subset of the input layer consisting of the most frequent words.

Feedforward NNLMs can be trained efficiently using back propagation algorithm [189], which is detailed in Chapter 3.2.2.

3.1.2 Recurrent Neural Network Language Model

As previously stated, feedforward NNLMs mitigate the data sparsity issue by projecting words into a low-dimension, continuous, space with a shared projection matrix. However, they still adopt an *n*-gram assumption where only the previous $n-1$ words are considered. Longer context information is discarded. To deal with this issue, recurrent neural network provides a feasible solution [153].

The structure of recurrent neural network based language models (RNNLMs) is illustrated in Figure 3.2. There are several differences compared to feedforward NNLMs. Only the previous word, instead of the previous $n-1$ words, is presented at the input layer. However, a recurrent connection between hidden and input layers is also added. In this way, RNNLMs [153] are able to represent the complete, non-truncated, history. In the input layer, a 1-of-K coding vector is again used for the input word w_{i-1}, similar to feedforward NNLMs, the 1-of-K coding word vector $\mathbf{r}^{(w_{i-1})}$ can be obtained according to Equation 3.2.1. The continuous vector $\mathbf{v}_{i-2}$ captures long term history from the start of a sequence via the recurrent connection. For an empty history, this is initialised, for example, to be a vector of 0.1. The probability of any given sentence $\mathcal{W}$ in RNNLMs can be written as,

$$\begin{aligned} P(\mathcal{W}) &= \prod_{i=1}^{N} P(w_i|w_{i-1},...w_1,w_0) \\ &\approx \prod_{i=1}^{K} P(w_i|w_{i-1},w_0^{i-2}) \\ &\approx \prod_{i=1}^{K} P(w_i|w_{i-1},\boldsymbol{v}_{i-2}) \\ &\approx \prod_{i=1}^{K} P(w_i|\boldsymbol{v}_{i-1}) \end{aligned} \tag{3.10}$$

It can be seen that the complete history of word w_i can be represented with two forms, one is the previous word w_{i-1} and a continuous history vector $\boldsymbol{v}_{i-2}$, the other is the continuous history vector $\boldsymbol{v}_{i-1}$.

The standard topology of a recurrent neural network language model consists of three layers. The full history vector, obtained by concatenating 1-of-K coding vector $\mathbf{r}^{(w_{i-1})}$ and recurrent history vector $\mathbf{v}_{i-2}$, is fed into the input layer. The hidden layer compresses the information from these two inputs and computes a new history representation $\mathbf{v}_{i-1}$ using a sigmoid activation to achieve non-linearity, as,

$$\mathbf{v}_{i-1} = \sigma(\mathbf{A}^\top \mathbf{r}^{(w_{i-1})} + \mathbf{B}^\top \mathbf{v}_{i-2}) \tag{3.11}$$

The history vector $\mathbf{v}_{i-1}$ is then passed to the output layer to produce the normalised RNNLM probabilities using a softmax activation as shown in Equation 3.9, as well as recursively fed back into the input layer as the "future" remaining history to compute the LM probability for the following word. The shared linear projection layer matrix $\mathbf{E}$ in feedforward NNLMs in Figure 3.1 is dropped in the RNNLM in this thesis. In the RNNLM, the linear projection layer and the hidden layer can be integrated into one matrix by simply multiplicating these two matrices as the input layer only contains the previous word. Hence, in this thesis, the projection layer for input layer is removed for RNNLMs and it doesn't affect performance[150]. For the training of recurrent neural network language models, back propagation through time (BPTT) is normally applied for optimisation, which is described in Chapter 3.2.

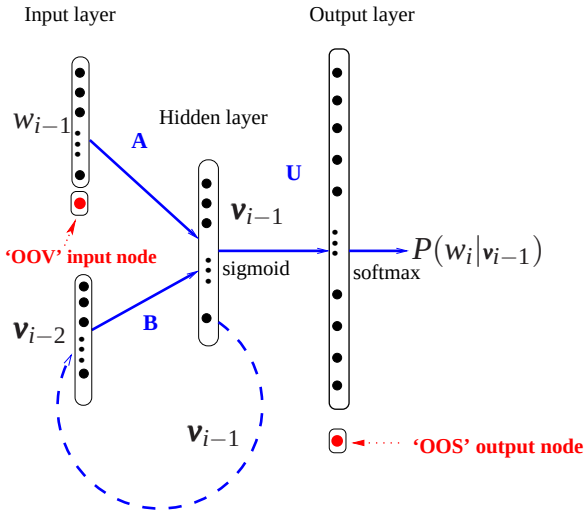


Fig. 3.2 *Recurrent neural network language model. The previous word w_{i-1} is projected into a low-dimension and continuous space via the projection layer, the complete history is modelled by a recurrent connection. The probability of word w_i* can be obtained from the output of softmax in the recurrent neural network.

RNNLMs handle the data sparsity and short term history issues by using a continuous word representation matrix $\mathbf{A}$ and recurrent connection matrix $\mathbf{B}$. Promising performance

has been reported on a range of tasks and applications [153, 154, 217, 248, 149, 32, 38]. Theoretically, the recurrent neural network can store the whole history information from the start of sentence. However, the gradient vanishes quickly during BPTT when simple sigmoid units are used in the hidden layer [13], which is discussed later in Chapter 3.2.3. Several amenable solutions are proposed to mitigate this, such as the use of Relu activation function [124]. A popular solution is to adopt long short term memory (LSTM) unit. LSTM unit is able to capture longer history than sigmoid, by introducing several gates to control the flow of information to overcome the gradient vanishing issue. Hence, it is used widely for recurrent neural networks to further improve performance.

3.1.3 Long Short Term Memory based RNNLM

The long short term memory (LSTM) network was proposed in 1997 [98]. The differences between LSTM based RNNLM and standard RNNLM lie in that the LSTM adopts a more complicated hidden unit rather than simple sigmoid. There are several variants for LSTM unit [84]. In this chapter, the most popular LSTM unit is described [81].

The central idea behind the LSTM is that the memory cell can maintain its state over time. Non-linear gating functions are added to control the information flow into and out of the LSTM unit. A typical structure of LSTM is shown in Figure 3.3. Three gating functions are introduced in the input, output and cell, which are called input gate, output gate and forget gate respectively. The input of the cell unit is scaled by the input gate. The input from the cell unit in the last step is scaled by the forget gate. The output of the cell unit, is scaled by the output gate. Peephole connections between the cell unit and the gates can be introduced into LSTM in [75] to learn the precise timing.

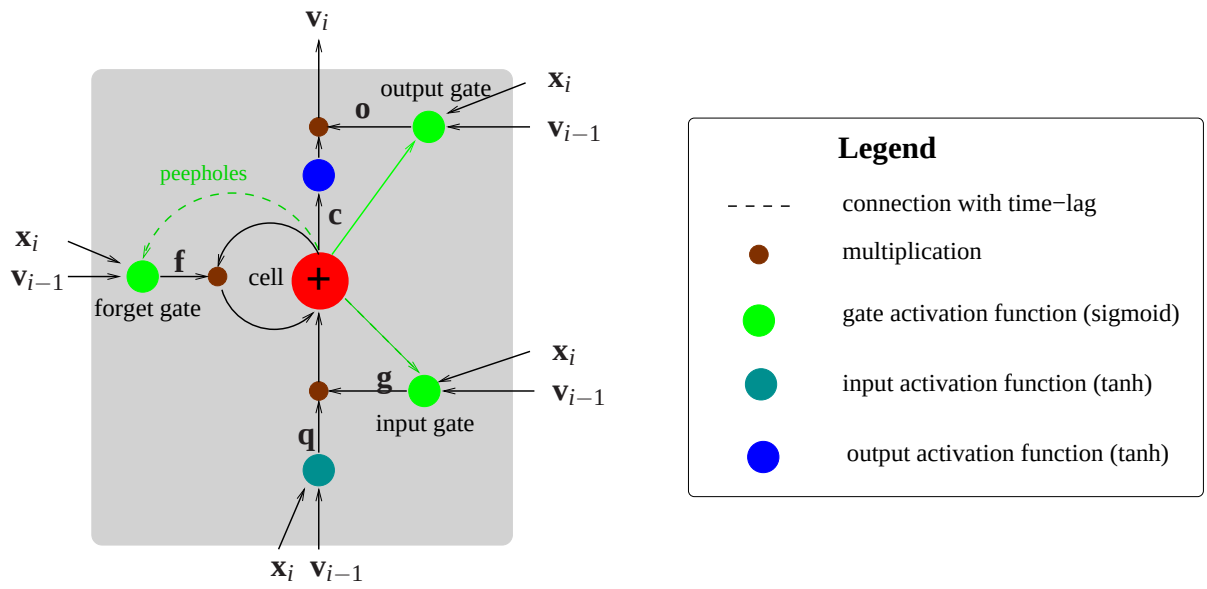


Fig. 3.3 *Long short-term memory unit. The gating functions, input, forget and output gates, are introduced into the model to control the signal flow.*

Normally, sigmoid function σ is used as the non-linear function for various gates. tanh function φ is chosen as non-linear function for the input and output of the unit block. The signal flow in the LSTM block is defined as

$$\begin{aligned}
\mathbf{q}_i &= \varphi(\mathbf{W}_z\mathbf{x}_i + R_z\mathbf{v}_{i-1} + \mathbf{b}_z) && \textit{block input} \\
\mathbf{g}_i &= \sigma(\mathbf{W}_g\mathbf{x}_i + R_g\mathbf{v}_{i-1} + \mathbf{t}_g \odot \mathbf{c}_{i-1} + \mathbf{b}_g && \textit{input gate} \\
\mathbf{f}_i &= \sigma(\mathbf{W}_f\mathbf{x}_i + R_f\mathbf{v}_{i-1} + \mathbf{t}_f \odot \mathbf{c}_{i-1} + \mathbf{b}_f && \textit{forget gate} \\
\mathbf{c}_i &= \mathbf{g}_i \odot \mathbf{q}_i + \mathbf{f}_i \odot \mathbf{c}_{i-1} && \textit{cell state} \\
\mathbf{o}_i &= \sigma(\mathbf{W}_o\mathbf{x}_i + R_o\mathbf{v}_{i-1} + \mathbf{t}_o \odot \mathbf{c}_i + \mathbf{b}_o) && \textit{output gate} \\
\mathbf{v}_i &= \mathbf{o}_i \odot \varphi(\mathbf{c}_i) && \textit{block output}
\end{aligned} \tag{3.12}$$

where $\odot$ denotes element-wise multiplication, and $\mathbf{t}_g$, $\mathbf{t}_f$ and $\mathbf{t}_o$ are the vectors for peephole connection to input, forget and output gates. The input of the LSTM network is $\mathbf{x}_t$ and $\mathbf{v}_{i-1}$, which is the input from previous layer and previous time slot. The output of LSTM network $\mathbf{v}_i$ will be used as input for the following prediction. Figure 3.4 gives an illustration of the LSTM based language model with a single LSTM layer. The training of LSTM based RNNLMs is the same as RNNLMs using back propagation through time. However, the number of parameters in LSTM based RNNLMs is much larger than simple RNNLMs with sigmoid. LSTM has been reported to produce better performance than a simple sigmoid unit in a range of tasks and applications including language models [220, 191, 221].

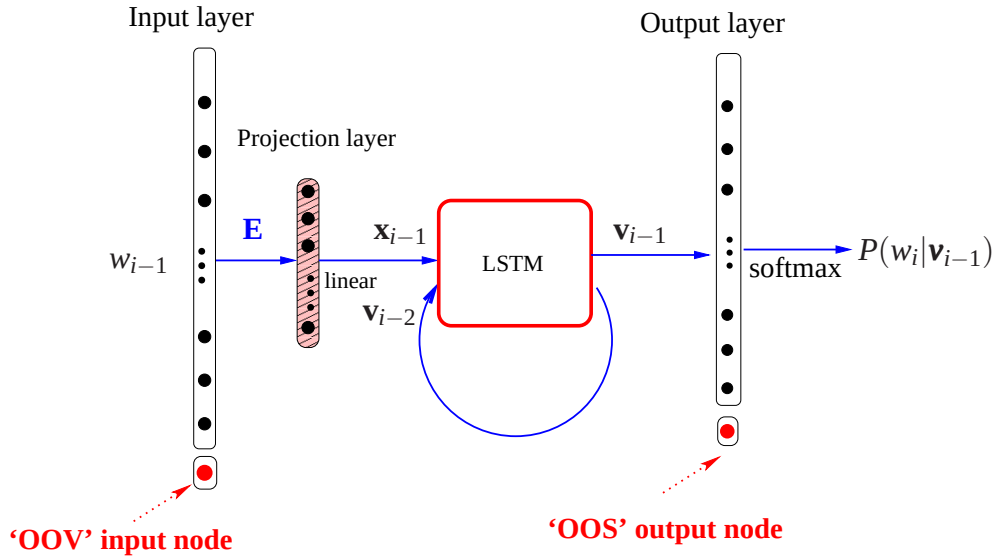


Fig. 3.4 *LSTM based language model with one LSTM layer.*

A similar machenism known as gated recurrent unit (GRU) [48] was also proposed recently using two gates (update gate and reset gate) to control the signal flow to avoid gradient vanishing. GRU was reported to give similar performance as LSTM on various tasks [48, 105]

It is worth pointing out that we didn't investigate LSTM and GRU activation based RNNLMs, we only looked into sigmoid activation based RNNLMs. However, the methods studied in the following sections (e.g. efficient training and evaluation, lattice rescoring, adaptation and interpolation) can be easily extended and applied to LSTM and GRU activation based RNNLMs.

3.2 Training of Neural Network based Language Model

In this section, the training of neural network based language model is described. Cross entropy is the standard objective function for the training of neural networks. Alternative objective functions will be discussed in Chapter 4. Back propagation and back propagation through time are adopted to optimise feedforward and recurrent neural networks respectively.

3.2.1 Cross Entropy

Cross entropy is a standard criterion used widely for training neural network language models. The cross entropy of the probability distributions from reference and neural network is minimised. Given one predicted word and the history h, the cross entropy criterion can be written as,

$$J^{\mathsf{CE}}(\boldsymbol{\theta}) = -\sum_{w_j \in \mathcal{V}^{out}} P_r(w_j|h) \log P(w_j|h) \tag{3.13}$$

where $\mathcal{V}^{out}$ is the vocabulary in the output layer and $\boldsymbol{\theta}$ is the set of model parameters (i.e. weight matrices and bias vectors) in the neural network to be optimised. $P_r(w_j|h)$ is the word probability in the reference for supervised training and only the probability of a single word is 1 and all other words are 0. Hence the cross entropy based objective function over the whole training corpus can be written as,

$$J^{\mathsf{CE}}(\boldsymbol{\theta}) \;=\; -\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|h_i) \tag{3.14}$$

where N is the number of training words. The optimisation of Equation 3.14 is equivalent to minimising the negative log-likelihood of NNLM probabilities over the training data.

3.2.2 Back Propagation

There are several approaches to minimise the cross entropy based objective function defined in Equation 3.14, for example, stochastic gradient descent (SGD) and Hessian free optimisation [146]. In this thesis, only the SGD is described and used. Back propagation is widely used as efficient SGD implementation in feedforward neural network.

Consider a feedforward neural network with L hidden layers, where the input and output of the lth layer (after sigmoid function) are $\mathbf{x}^{(l)}$ and $\mathbf{v}^{(l)}$; The weight matrix in layer l is

denoted as $W^{(l)}$ and the bias vector is $\mathbf{b}^{(l)}$. The forward process can be written as,

$$\begin{aligned}\mathbf{x}^{(l)} &= \mathbf{W}^{(l-1)^\top}\mathbf{v}^{(l-1)} + \mathbf{b}^{(l-1)} \\ \mathbf{v}^{(l)} &= \sigma(\mathbf{x}^{(l)})\end{aligned} \tag{3.15}$$

again, where σ is the sigmoid function as,

$$\sigma(v_j) = \frac{1}{1+\exp(-v_j)} \tag{3.16}$$

The output of kth node in the output layer is,

$$y_k = \frac{\exp(x_k^{(L)})}{\sum_{j=1}^{|\mathcal{V}^{out}|}\exp(x_j^{(L)})} \tag{3.17}$$

where $|\mathcal{V}^{out}|$ is the size of output layer.

Given the objective function $J^{\mathsf{CE}}(\boldsymbol{\theta})$, the error signal in the lth layer $\mathbf{g}^{(l)}$ during back propagation is written as,

$$\mathbf{g}^{(l)} = \frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \mathbf{x}^{(l)}} \tag{3.18}$$

In the output layer, the error signal in the kth node $g_k^{(L)}$ is,

$$g_k^{(L)} = \delta(k_r, k) - y_k \tag{3.19}$$

where δ is the Dirac delta function and k_r is the reference label. The error in lth layer can be derived using the following form,

$$g_j^{(l-1)} = \frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial x_j^{(l-1)}} = \sum_t \frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial g_t^{(l)}} \frac{\partial g_t^{(l)}}{\partial v_j^{(l-1)}} \frac{\partial v_j^{(l-1)}}{\partial x_j^{(l-1)}} \tag{3.20}$$

It can be written in the vector form as,

$$\mathbf{g}^{(l-1)} = \mathbf{W}^{(l)^\top}\mathbf{g}^{(l)} \odot \sigma'(\mathbf{x}^{(l-1)}) \tag{3.21}$$

where $\odot$ denotes the element-wise multiplication. The gradient of weight $\omega_{jk}^{(l)}$ connecting the jth node in $l-1$th layer and kth node in lth layer can be expressed as,

$$\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \omega_{jk}^{(l)}} = \frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial x_k^{(l)}} \frac{\partial x_k^{(l)}}{\partial \omega_{jk}^{(l)}} = g_k^{(l)} v_j^{(l-1)} \tag{3.22}$$

It can be again written with vetor and matrix form as,

$$\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \mathbf{W}^{(l)}} = \mathbf{v}^{(l-1)}\mathbf{g}^{(l)^{\top}} \tag{3.23}$$

Equation 3.21 and 3.23 can be used to calculate the error signal and gradient weight from layer $L-1$, layer $L-2$, to layer 1 recursively with a backward direction, so this algorithm is called back propagation in the literature [189].

The above derivation on back propagation algorithm can be applied in all feedforward neural networks. Neural network language models are a special type of neural network where the input of the neural network consists of 1-of-K coding vector where only a single node is 1 and all the others are 0. Hence, computation can be reduced significantly in the input layer. Only the weight vector associated with the input word needs to be computed and updated.

3.2.3 Back Propagation Through Time

Training of neural networks with recurrent connection (e.g. standard RNNLM and LSTM based RNNLM), requires an extension to the standard back propagation, back propagation through time [190]. In addition to the error from the upper layer in standard back propagation, the error is also propagated through the recurrent connection. The recurrent connection can be unfolded in time as shown in Figure 3.5. The network can also be viewed as a deep network in time, with shared weight matrices **A** and **B**.

However, it is computationally expensive to unfold the RNN to the begin of sentence for every prediction. Assuming the computational complexity of the back propagation and update of A and B is C, for a sentence with N words, the whole computation happening on the hidden layer for BPTT is

$$(1+2+...+N)\times C = \frac{N(N+1)}{2}C \tag{3.24}$$

In order to reduce computation, a truncated version of BPTT [236, 238, 153] was adopted in many works. Instead of tracing back to the start of sentence, only a fixed and finite step T is back propagated, e.g. 5. The computation complexity for updating the hidden layer turns out to be $N \times T \times C$. The computation can be reduced significantly compared to Equation 3.24 when the length sentence is long enough ($N >> T$).

$\boldsymbol{v}_0$ is the initial history vector at the sentence start as discussed before. During the computation of gradient, the error signal is back propagated as the flow of grey arrows shown in Figure 3.5. The gradients are average over time due to the shared weight matrices **A** and **B**. This sharing effectively limits the number of model parameters.

In RNNLMs with sigmoid activation function as discussed in Chapter 3.1.2, the error is back propagated using Equation 3.21, which can be rewritten as below,

$$\mathbf{g}^{(l-1)} = \mathbf{W}^{(l)^{\top}}\mathbf{g}^{(l)} \odot \sigma'(\mathbf{x}^{(l-1)}) \tag{3.25}$$

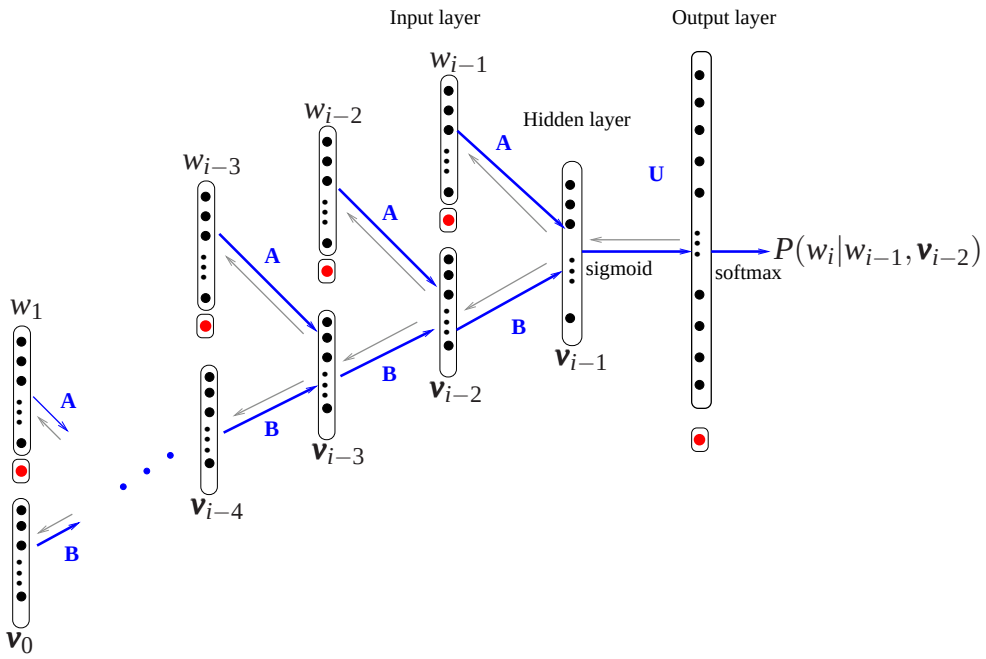


Fig. 3.5 *Back propagation through time for Recurrent NNLM. The RNNLM can be unfolded through time and the projection layer A and recurrent layer B are shared in different time steps.*

σ is sigmoid function and its derivation to input $\mathbf{x}^{(l-1)}$ can be written as,

$$\sigma'(\mathbf{x}^{(l-1)}) = \mathbf{x}^{(l-1)} \odot (1 - \mathbf{x}^{(l-1)}) \tag{3.26}$$

The output of sigmoid function is positive and smaller than 1 according to the definition in Equation 3.7. Hence, the error signal is decayed exponentially and close to 0 after several back propagations. The gradient is calculated according to Equation 3.23 and also close to 0 [13]. To address this issue, LSTM network introduces a gating function, which helps to avoid the gradient decay existing in standard RNNLMs. BPTT can be applied in the training of LSTM based RNNLM as well in a similar way.

3.3 Application of RNNLMs for Speech Recognition

In many speech recognition systems, RNNLMs are normally trained on a small amount of in-domain data (e.g. acoustic model transcription) with a small hidden layer size (e.g. 200) and output layer (e.g. 20K). The standard *n*-gram LMs are first used in the decoding and lattices are generated. Then, the N-best lists are extracted from the lattices and rescored by combining the RNNLMs and *n*-gram LMs. Linear interpolation is the most popular approach to combine RNNLMs and *n*-gram LMs. Despite the success achieved for RNNLMs in speech recognition [217, 151, 235, 123, 38], there are still several issues to be addressed and aspects to be explored.

The training of RNNLMs can be computationally heavy, especially when a large vocabulary in the output layer is applied. In order to reduce the computational cost, in previous work, a shortlist [196, 62] on the output layer limited to the most frequent words was used. Class based output layer [154, 164] was also proposed for neural network language models. However, it is difficult to parallel the training of RNNLMs and slow to train large RNNLM model on the corpus with a large amount of words. The efficient training and inference of RNNLMs will be discussed in Chapter 4.

As mentioned above, N-best rescoring ,instead of lattice rescoring, is normally used for RNNLMs in speech recognition. The N-best lists only contain a small subset of hypotheses in the lattice and largely limit the search space for RNNLMs. It is of practical value for RNNLMs to support lattice rescoring and be able to generate lattices, Lattices are useful for many downstream applications, such as consensus decoding and keyword spotting. Lattice rescoring methods using RNNLMs will be described in Chapter 5.

In most speech recognition systems, RNNLMs are applied without adaptation. Similar to the acoustic model adaptation, the mismatch also exists in language model. As described in Chapter 2.3.6, there are many works have been done in standard *n*-gram LMs [11]. It is also important to investigate the adaptation of RNNLMs for speech recognition, which is discussed in Chapter 6.

In state-of-the-art ASR systems, RNNLMs are often linearly interpolated with *n*-gram LMs to obtain both a good context coverage and strong generalisation [153, 196, 172, 131]. The interpolated LM probability is given by,

$$P(w|h) \;=\; \lambda P_{\mathsf{NG}}(w|h) + (1-\lambda) P_{\mathsf{NN}}(w|h) \tag{3.27}$$

where λ is the interpolation weight of *n*-gram LM, and λ can be optimised via EM algorithm on a held-out set as discussed in Chapter 2.3.3. In the above interpolation, the probability mass of OOS words assigned by the RNNLM component is re-distributed with equal probabilities among all OOS words to guarantee a valid probability. A better but more complicated way is to use unigram or higher order *n*-gram LM probability for rescaling [172]. The interpolation between RNNLMs and *n*-gram LMs is studied in Chapter 7.

3.4 Summary

Neural network based language models (NNLMs) have been discussed in this chapter. Three types of neural networks used to construct language models in the literature are introduced, including feedforward, recurrent and long short term memory (LSTM) recurrent neural network language models. Standard *n*-gram LMs have issues with data sparsity and *n*-gram history. Feedforward NNLMs mitigate the data sparsity issue by representing each word with a low-dimension and continuous vector. Recurrent NNLMs are able to model the long term history via the recurrent connection. LSTM based RNNLMs are more capable of modelling longer history by introducing gating functions to control information flow. The training of NNLMs are detailed, such as back propagation algorithm for feedforward neu-

ral networks and back propagation through time for recurrent neural networks. Finally, the application of NNLMs in speech recognition is also discussed in this chapter.

Chapter 4

Efficient Training and Inference of RNNLMs

One practical issue associated with RNNLMs is the computational cost incurred in model training and inference. The training time increases linearly with the amount of training data. Additionally as the model size grows, especially when a large output vocabulary is chosen, the training and decoding time increase. This limits the potential applications of RNNLMs especially in the scenario where there are large quantities of data available. Hence, it is of great practical value to develop techniques for rapid RNNLM training and efficient inference. Unlike most previous work, where RNNLMs were trained on CPUs with a factorised output layer, in this chapter we explore the efficient RNNLM training on GPU with full output layer. In order to facilitate bunch [1] mode during RNNLM training, a novel data structure, sentence splice, is proposed to minimise redundant computation. In addition to the conventional cross entropy based training, two improved training criteria are investigated for fast RNNLM training and inference [46].

This chapter is organised as follows. In Chapter 4.1 recurrent neural network LMs are reviewed and two RNNLM architectures (i.e. full output and class output layer RNNLMs) are presented. In addition to the conventional cross entropy criterion, variance regularisation and noise contrastive estimation are introduced for training of RNNLMs in Chapter 4.2. In Chapter 4.3, the computational complexities among different model structures and training criteria are discussed. In order to apply bunch (i.e. minibatch) based training for RNNLMs, a novel spliced sentence bunch mode parallelisation algorithm for RNNLM training is proposed and its GPU based implementation described in Chapter 4.4. Pipelined RNNLM training is discussed in Chapter 4.5 to further speed up training by using multiple GPUs. In Chapter 4.6 the performance of the proposed F-RNNLMs training and efficiency improving techniques are evaluated on a large vocabulary conversational telephone speech transcription system and Google's one billion word benchmark task. Finally, conclusions are drawn in Chapter 4.7 .

[1] also known as minibatch in the literature

4.1 Recurrent Neural Network LMs Structures

There are two types of structure used for RNNLMs in the literature [153, 154]. The difference lies in the output layer. The two types are called full output and class output layer based RNNLMs. Class output layer based RNNLMs [154] were originally introduced to reduce computational load at both training and test time by using a factorised output layer. Hence, it is very popular and used widely in previous work [154, 257, 26]. Conventionally a direct implementation of RNNLM yields a full output layer RNNLM.

Based on the discussion in Chapter 3, the computation of probability over a sentence given RNNLMs can be written as,

$$\begin{aligned} P(\mathcal{W}) &= \prod_{i=1}^{N} P(w_i|w_{i-1},...,w_1,w_0) \\ &\approx \prod_{i=1}^{N} P(w_i|w_{i-1},\boldsymbol{v}_{i-2}) \end{aligned} \tag{4.1}$$

where N is the number of words, and $\boldsymbol{v}_{i-2}$ is the history vector representing previous words $\{w_0, w_1, ..., w_{i-2}\}$.

4.1.1 Full output layer based RNNLMs (F-RNNLMs)

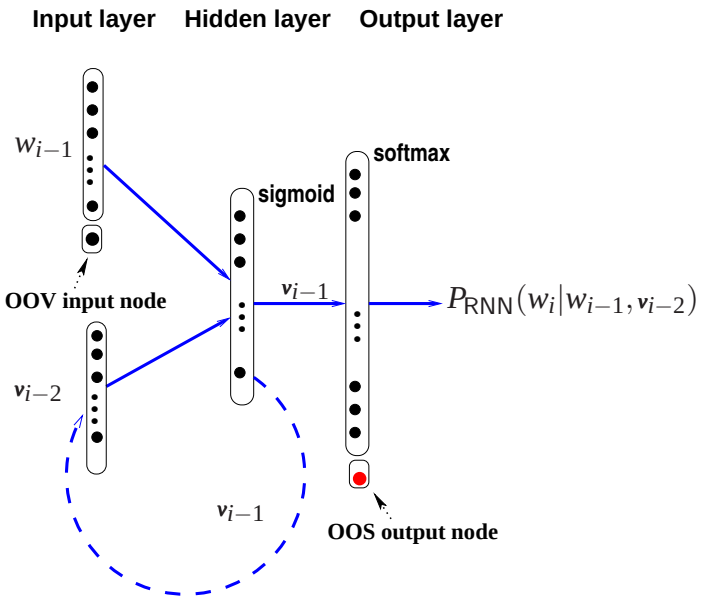


Fig. 4.1 *An example RNNLM with an full output layer, An out-of-vocabulary (OOV) node is added in the input layer and out-of-shortlist (OOS) nodes is added in the output layer to model unseen words.*

The direct application of RNNs to language modelling is illustrated in Figure 4.1. The topology of the recurrent neural network used to compute LM probabilities $P_{\text{RNN}}(w_i|w_{i-1},\boldsymbol{v}_{i-2})$ consists of three layers. The history vector of w_i, obtained by concatenating w_{i-1} and $\boldsymbol{v}_{i-2}$, is fed into the input layer. The hidden layer compresses the information of these two inputs and computes a new representation $\boldsymbol{v}_{i-1}$ using sigmoid activation function σ to achieve non-linearity. The sigmoid function is expressed as,

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{4.2}$$

In order to compute the word predicted probability $P_{RNN}(w_i|w_{i-1},\boldsymbol{v}_{i-2})$, softmax function is used in the output layer for normalisation, which is given as,

$$P(w_i|w_{i-1},\boldsymbol{v}_{i-2}) = \frac{\exp(y_{w_i})}{\sum_{j=1}^{|\mathcal{V}^{out}|}\exp(y_{w_j})} \tag{4.3}$$

where y_{w_j} is the output for w_j and $|\mathcal{V}^{out}|$ is the output layer size.

The output layer of RNNLMs is normally quite large, which causes heavy computation during the probability estimation. To reduce the computational cost, a shortlist [196, 62] based output layer vocabulary limited to the most frequent words is used. A similar approach may also be used at the input layer. To reduce the bias to in-shortlist words during RNNLM training and improve robustness, an additional node is added at the output layer to model the probability mass of out-of-shortlist (OOS) words [172, 131, 142].

4.1.2 Class Based RNNLMs (C-RNNLMs)

As stated above, training F-RNNLMs is computationally expensive, and the major part of cost is incurred in the output layer. Existing techniques have focused on class based RNNLMs (C-RNNLMs), an architecture with a class based factorised output layer [154, 164]. An example C-RNNLM is illustrated in Figure 4.2. There are two matrices in the output layer, which are class and word output layer respectively. Each word in the output layer vocabulary is attributed to a unique class and each class contains a group of words, which is a subset of the output vocabulary. The weight matrix of class output layer is randomly initialised in the same way as other weight matrices. The predicted LM probability assigned to a word is factorised into two individual terms,

$$P_{RNN}(w_i|h_i) = P(w_i|w_{i-1},\boldsymbol{v}_{i-2}) = P(w_i|c_i,\boldsymbol{v}_{i-1})P(c_i|\boldsymbol{v}_{i-1}) \tag{4.4}$$

where h_i and $\boldsymbol{v}_{i-1} = \{w_{i-1},\boldsymbol{v}_{i-2}\}$ both represent the history of word w_i.

The class probability $P(c_i|\boldsymbol{v}_{i-1})$ is calculated first, and $P(w_i|c_i,\boldsymbol{v}_{i-1})$ then calculates the word probability w_i within class c_i. The calculation of word probability $P(w_i|c_i,\boldsymbol{v}_{i-1})$ is based on a small subset of words from the same class, and the number of classes is normally significantly smaller than the full output layer size. Class output layer based RNNLM provides significant speedup compared to full output layer both in train and decode stages [154].

When further combined with parallelised model training [205] and multi-stage classing at the output layer [103], training time speedup to 10 fold were reported compared to previous research for C-RNNLMs. A special case of C-RNNLM with a single class is equivalent to a traditional, full output layer based F-RNNLM.

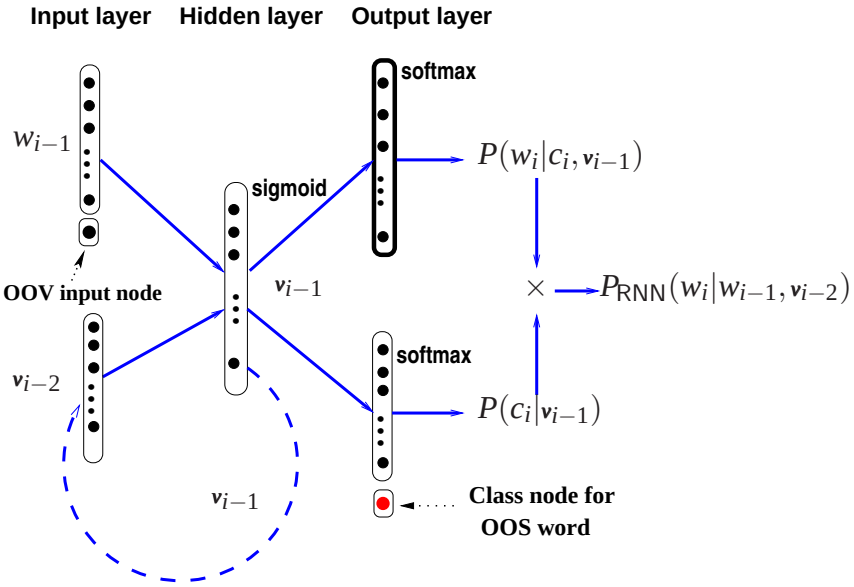


Fig. 4.2 *An example RNNLM with a class-based output layer. An out-of-vocabulary (OOV) node is added in the input layer and out-of-shortlist (OOS) nodes is added in the output layer to model unseen words.*

The speedup of C-RNNLMs is based on the assumption that hidden layer size H is significantly smaller than output layer size $V = |\mathcal{V}^{out}|$. However, there are several issues associated with these approaches. First, the use of class based output layer limits the potential speedup from bunch[2] mode training parallelisation [217]. Words from the same bunch may be from different classes, which requires the call of different submatrices in the word output layer. This complicates the implementation, especially when GPUs are used since GPU is more suitable and better optimised for regular matrix operation. Second, the underlying word to class assignment scheme at the output layer may also affect the resulting C-RNNLM's performance [154, 257, 43, 128]. Finally, most previous work focus on CPU based speedup techniques [154, 217, 205, 103]. Hence, it is preferable to also exploit the parallelisation power of GPUs.

It is worth noting that the output layer in the class based RNNLM shown in Figure 4.2 can be viewed as a two-layer hierarchical output layer. A more genral hierarchical output layer gives more speedup since the computation could be further reduced, which gives a computation complexity of $O(\log(V) * H * N)$ in the output layer. This is out of the scope of this thesis and more details about the hierarchical output layer can be found in [164].

[2]It is also sometimes referred as "minibatch" in literature [37, 199]. For clarity, the term "bunch" is used throughout this thesis.

4.2 RNNLM Training Criteria

Cross entropy (CE) is the conventional criterion used for RNNLM training as discussed in Chapter 3.2.1. The log likelihood is maximised during training. However, for RNNLMs, it requires the calculation of the normalised probability both in the train and test time, which is computationally heavy. To improve efficiency, two alternative criteria will be discussed. They are variance regularisation [208] and noise contrastive estimation respectively [159]. Conventional F-RNNLMs are chosen for discussion in this section. These criteria can also be applied to C-RNNLMs.

4.2.1 Cross Entropy

Conventional RNNLM training [153] was discussed in Chapter 3.2.1, which aims to maximise the log-likelihood, or equivalently minimise the cross entropy (CE) measure of the training data. For a train corpus containing N words, the objective function is given by,

$$J^{\mathsf{CE}}(\boldsymbol{\theta}) \;=\; -\frac{1}{N}\sum_{i=1}^{N}\ln P_{\mathsf{RNN}}(w_i|h_i) \tag{4.5}$$

where

$$P_{\mathsf{RNN}}(w_i|h_i) \;=\; \frac{\exp\left(\boldsymbol{\theta}_i^{\top}\boldsymbol{v}_{i-1}\right)}{\sum_{j=1}^{|\mathcal{V}^{out}|}\exp\left(\boldsymbol{\theta}_j^{\top}\boldsymbol{v}_{i-1}\right)} \;=\; \frac{\exp\left(\boldsymbol{\theta}_i^{\top}\boldsymbol{v}_{i-1}\right)}{Z(h_i)} \tag{4.6}$$

is the probability of word w_i given history h_i. $\boldsymbol{\theta}_i$ is the weight vector associated with word w_i at the output layer. $\boldsymbol{v}_{i-1}$ is the hidden history vector computed, and $|\mathcal{V}^{out}|$ is the size of output layer vocabulary. The gradient used in the conventional CE based training for RNNLMs is,

$$\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = -\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial\left(\boldsymbol{\theta}_i^{\top}\boldsymbol{v}_{i-1}\right)}{\partial\boldsymbol{\theta}} - \sum_{j=1}^{|\mathcal{V}^{out}|}P_{\mathsf{RNN}}(w_j|h_i)\frac{\partial\left(\boldsymbol{\theta}_j^{\top}\boldsymbol{v}_{i-1}\right)}{\partial\boldsymbol{\theta}}\right) \tag{4.7}$$

The denominator term $Z(h_i)$ in Equation 4.6 performs a normalisation over the full output layer, which is given as,

$$Z(h_i) = \sum_{j=1}^{|\mathcal{V}^{out}|}\exp\left(\boldsymbol{\theta}_j^{\top}\boldsymbol{v}_{i-1}\right) \tag{4.8}$$

This operation is computationally expensive when computing the RNNLM probabilities during both test time and CE based training when the gradient information of Equation 4.7 is calculated. As discussed in Chapter 4.4, the efficient bunch mode GPU based parallelisation with sentence splicing is used to improve the speed of conventional CE training.

4.2.2 Variance Regularisation

In many applications, RNNLMs are required to be efficiently evaluated in test time. The softmax calculation is computationally heavy for full output layer RNNLMs given the large output layer size, especially when CPUs are used. One technique that can be used to improve the testing speed is introducing the variance of the normalisation term into the conventional cross entropy based objective function of Equation 4.6. In previous research, variance regularisation (VR) (or called self-normalisation) has been applied to training of feedforward NNLMs and class based RNNLMs [61, 208, 209]. By explicitly minimising the variance of the softmax normalisation term during training, the normalisation term at the output layer can be viewed as constant and ignored during test time, thus significant improvement in speed is achieved. The conventional CE objective function of Equation 4.5 could be written as below in VR based training,

$$J^{\mathsf{VR}}(\boldsymbol{\theta}) \;=\; J^{\mathsf{CE}}(\boldsymbol{\theta}) + \frac{\gamma}{2N}\sum_{i=1}^{N}\left(\ln Z(h_i) - \overline{\ln Z}\right)^2 \tag{4.9}$$

where $J^{\mathsf{CE}}(\boldsymbol{\theta})$ and $Z(h_i)$ are the cross entropy based training criterion and the softmax normalisation term associated with a history h_i in Equation 4.6 respectively, and $\overline{\ln Z}$ is the mean of the log scale normalisation term computed as,

$$\overline{\ln Z} = \frac{1}{N}\sum_{i=1}^{N}(\ln Z(h_i)) \tag{4.10}$$

Although according to this equation, $\overline{\ln Z}$ is a function of model parameter in RNNLMs, $\overline{\ln Z}$ is viewed as constant and fixed for each minibatch as an approximation. γ is a tunable parameter to adjust the contribution of the variance regularisation term. Directly maximising the above objective function in Equation 4.9 could explicitly minimise the variance of the softmax normalisation term. The gradient used in the variance regularisation based training is given by,

$$\frac{\partial J^{\mathsf{VR}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\gamma}{N}\sum_{i=1}^{N}\left((\ln Z(h_i) - \overline{\ln Z}) \times \sum_{j=1}^{|\mathcal{V}^{out}|} P_{\mathsf{RNN}}(w_j|h_i)\frac{\partial(\boldsymbol{\theta}_j^{\top}\boldsymbol{v}_{i-1})}{\partial \boldsymbol{\theta}}\right) \tag{4.11}$$

where $\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ is the CE gradient given in Equation 4.7, and $P_{\mathsf{RNN}}(\cdot|h_i)$ is the conventional RNNLM probabilities computed from Equation 4.6. From Equation 4.11, the computational load required for the update of VR is the same as CE since the normalisation term $Z(h_i)$ can be cached during the softmax calculation. Hence, the computation during training in VR is the same as CE.

However, in test time, variance regularisation allows a history independent, constant softmax normalisation term to be used. The RNNLM probabilities are thus approximated as,

$$P^{\mathsf{VR}}_{\mathsf{RNN}}(w_i|h_i) \;\approx\; \frac{\exp(\boldsymbol{\theta}_i^{\top}\boldsymbol{v}_{i-1})}{Z} = \frac{\exp(\boldsymbol{\theta}_i^{\top}\boldsymbol{v}_{i-1})}{\exp(\overline{\ln Z})}. \tag{4.12}$$

where

$$Z = \exp(\overline{\ln Z}) \tag{4.13}$$

is the constant normalisation term obtained during training. The computation of $\overline{\ln Z}$ is given in Equation 4.10. This significantly reduces the computation at the output layer as the normalisation is no longer required. This means that the computational load is less sensitive to the size of output layer vocabulary $|\mathcal{V}^{out}|$, a maximum $|\mathcal{V}^{out}|$ times speedup at the output layer can be achieved in test time.

In this thesis, variance regularisation based training is used to improve the inference efficiency [43, 201] and can be integrated with the bunch mode based training introduced in Chapter 4.4. In contrast to setting the mean of log normalisation term $\overline{\ln Z}$ to zero as previous research for C-RNNLMs [208] and feedforward NNLM in [61], it is found that calculating $\overline{\ln Z}$ separately for individual bunches gave improved convergence speed and stability in F-RNNLM training. During test time, the approximated normalisation term Z is computed on the validation set, and remains fixed during performance evaluation for all experiments.

4.2.3 Noise Contrastive Estimation

The explicit computation of normalisation term required at the output layer significantly impacts both the training and testing speed of RNNLMs. Variance regularisation can significantly reduce the associated computation during test time. However, the explicit computation of this normalisation term is still required in training and used to compute the variance regularised gradient information in Equation 4.11. This train speed of variance regularisation is the same as cross entropy based training and is still limited when a large output layer is used. A more general solution to this problem is to use techniques that remove the need to compute such normalisation term in both training and testing. One such technique investigated in this thesis is based on noise contrastive estimation (NCE) [44, 88, 239, 201].

NCE provides an alternative solution to estimate normalised statistical models when the exact computation of the normalisation term is either computationally impossible or highly expensive to perform; for example, in feedforward and recurrent NNLMs, when a large output layer vocabulary is used. The central idea of NCE is to perform nonlinear logistic regression classification to discriminate between the observed data and some artificially generated noise data. The variance of the normalisation term is minimised implicitly during training due to the normalisation of noise distribution. Hence, it allows normalised statistical models, for example, NNLMs, to use "unnormalised" probabilities without explicitly computing the normalisation term during both training and testing. In common with the use of a class based output layer, the NCE algorithm presents a dual purpose solution to improve both the training and inference efficiency for RNNLMs.

In the NCE training of RNNLMs, for a given full history context h, data samples are generated from a mixture of two distributions: the NCE estimated RNNLM distribution $P_{\mathsf{RNN}}(\cdot|h)$, and some known noise distribution $P_n(\cdot|h)$, such as uniform distribution or unigram distribution. Assuming the noise samples are k times more frequent than true RNNLM

data samples, the distribution of data could be described as,

$$\frac{1}{k+1}P_{\mathsf{RNN}}(\cdot|h)+\frac{k}{k+1}P_n(\cdot|h). \tag{4.14}$$

Given history h, The posterior probability of word w is generated from the RNNLM is,

$$P(w\in D|w,h)=\frac{P_{\mathsf{RNN}}(w|h)}{P_{\mathsf{RNN}}(w|h)+kP_n(w|h)} \tag{4.15}$$

The posterior probability of word w is generated from a noise distribution is,

$$P(w\in N|w,h)=\frac{kP_n(w|h)}{P_{\mathsf{RNN}}(w|h)+kP_n(w|h)} \tag{4.16}$$

where $w\in D$ and $w\in N$ indicate the word w is generated by data and noise distribution respectively. In NCE training, for each train sample w_i and its history h_i, k noise samples $\check{w}_{i,j}(j=1,2,...k)$ are randomly sampled from a specified noise distribution (e.g. unigram distribution). The objective function is to minimise the log negative posterior probabilities of all samples, which is given as,

$$J^{\mathsf{NCE}}(\boldsymbol{\theta})=-\frac{1}{N}\sum_{i=1}^{N}\Big(\ln P(w_i\in D|w_i,h_i)+\sum_{j=1}^{k}\ln P(\check{w}_{i,j}\in N|\check{w}_{i,j},h_i)\Big) \tag{4.17}$$

The derivation of the gradient in the above equation could be found in Appendix A, which is computed as,

$$\begin{aligned}\frac{\partial J^{\mathsf{NCE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}= &-\frac{1}{N}\sum_{i=1}^{N}\Big(P(w_i\in N|w_i,h_i)\frac{\partial\ln P_{\mathsf{RNN}}(w_i|h_i)}{\partial\boldsymbol{\theta}}\\ &-\sum_{j=1}^{k}P(\check{w}_{i,j}\in D|\check{w}_{i,j},h_i)\frac{\partial\ln P_{\mathsf{RNN}}(\check{w}_{i,j}|h_i)}{\partial\boldsymbol{\theta}}\Big)\end{aligned} \tag{4.18}$$

The NCE trained RNNLM distribution is given by,

$$P_{\mathsf{RNN}}(w_i|h_i)\approx\frac{\exp\left(\boldsymbol{\theta}_i^{\top}\boldsymbol{v}_{i-1}\right)}{Z} \tag{4.19}$$

NCE training learns a constant, history context independent normalisation term Z, in contrast to the explicitly normalised RNNLM distribution that used during CE and variance regularisation training [3]. The normalisation term Z in NCE training is a constant and similar to that defined in Equation 4.13 for variance regularisation. This crucial feature not only allows the resulting RNNLM to learn the desired sum-to-one constraint of conventional CE

[3] A more general case of NCE training also allows the normalisation term to vary across different histories, thus incurring the same cost as in conventional CE based training [88].

estimated RNNLMs, but also to be efficiently computed during both training and test time without requiring explicit computation of the softmax normalisation term at the output layer.

Figure 4.3 gives an example of noise samples generated during NCE training. The unigram distribution obtained from the training corpus is used as noise distribution. For a given sentence "eating at home", sentence start <s> and sentence end </s> are added. For each predicted word in the sentence, 10 "noisy" words are randomly sampled from the unigram distribution. NCE training aims to discriminate the reference sentence from training data from noisy sentence generated from noise distribution (unigram distribution here).

Train sentence	<S>	EATING	AT	HOME	</S>
Noise Samples		AND	THIS	SHOULD	THAT'S
		HOUSE	AT	THINK	</s>
		PARENTS	I	SHOULD'VE	VERSUS
		THERE	ACTUALLY	JUST	SHE
		ALL	DOWN	WITH	TAMPA
		HOUR	ATTACK	</s>	THAT
		I	YEAH	TOWERS	SO
		EVERYTHING	BIG	BUT	HIGH
		THAT'S	</s>	THERE	STARTED
		OUT	I	</s>	AND

Fig. 4.3 *Noise samples generated for sentence "<s> eating at home </s>" by a unigram noise model for NCE training.*

The NCE objective function in Equation 4.17 is optimised on the training set and cross entropy is computed on the validation set with the normalised RNNLM probabilities shown in Equation 4.6. The cross entropy in validation set is used to control the learning rate. There are a number of parameters that need to be appropriately set. First, a noise distribution is required to generate noise samples. As suggested in earlier research [159, 230], a context independent unigram LM distribution is used to draw the noise samples. Second, the setting of k controls the bias towards the characteristics of the noise distribution. It also balances the trade-off between training efficiency and performance. For each target word w, a total of k noise samples are sampled independently from the noise distribution. It is worth noting that the noise sample could be the predicted word and the same noise sample may appear more than once. Finally, NCE training also requires a constant normalisation term Z as in Equation 4.19. In previous research on NCE training of log-bilinear LMs [159] and feed-forward NNLMs [230], the constant normalisation term was set as $\overline{\ln Z} = 0$. For RNNLMs, an empirically adjusted setting of $\overline{\ln Z} = 9$, which is close to the mean of the log scale normalisation term computed on the training data using a randomly initialised RNNLM [4]. This setting was found to give a good balance between convergence speed and performance and used in all experiments.

[4]Other values were also tried, e.g. 6, 7, 8, 10. They gave similar results. But a value of 0 hurted performance.

The main advantages of RNNLMs training with NCE are summarised below. First, the computational load in the output layer is reduced dramatically as it only needs to considerate k noise samples and target word, instead of the whole output layer. Compared with the CE based training gradient given in Equation 4.5, NCE gradient calculation in Equation 4.18 is $\frac{|\mathcal{V}^{out}|}{k+1}$ times faster. Second, the train speed is less sensitive to output layer size, which allows RNNLMs with larger vocabulary to be trained. Finally, the normalisation term can be approximated as constant during NCE training. This can avoid the re-computation of the normalisation term for different histories, therefore allows the normalised RNNLM probabilities to calculated in test time with the same efficiency as unnormalised probabilities. In common with variance regularisation based training, a $|\mathcal{V}^{out}|$ times speedup at the output layer during test time can thus be achieved.

4.3 Computation Complexity Analysis

In previous sections, two different RNNLM structures, i.e. full and class output layer RNNLMs, were presented, and three train criteria were described for efficient training and inference. In this section, a quantitive analysis is given to illustrate the computation load associated with model structure and train criteria.

Assuming that RNNLM contains a single hidden layer with H hidden nodes, and the output layer size is $V = |\mathcal{V}^{out}|$, when cross entropy is used as objective function, in full output layer based RNNLM, for the prediction of each word, the computational complexity during forward pass is proportional to,

$$(H+1) \times H + H \times V, \tag{4.20}$$

The non-linear computation in hidden layer and output layer is not counted in this computation analysis as it is proportional to H or V. During update, when the truncated back propagation through time (BPTT) [236] is applied, the computational complexity is proportional to,

$$(H+1) \times H \times \tau + H \times V, \tag{4.21}$$

where τ is BPTT step for each word. Normally, the output layer size V is significantly larger than hidden layer size H (i.e. $V >> H$). Under this assumption , the computational complexity over one epoch is propotional to $O(H \times V \times N)$, which is linearly related to hidden layer size H, output layer size V and number of training words N.

While for class based RNNLMs with C classes in the output layer, where each class has $\frac{V}{C}$ words on average, the computational complexity during forward for each training word is proportional to,

$$(H+1) \times H + (C + \frac{V}{C}) \times H \tag{4.22}$$

Table 4.1 Computational complexities for each RNNLM forward in the output layer during training and testing using different model structures and training criteria.

	CE		VR		NCE	
	full	class	full	class	full	class
Train	$H \times V$	$(C + \frac{V}{C}) \times H$	$H \times V$	$(C + \frac{V}{C}) \times H$	$H \times k$	$H \times k$
Test				H	H	H

The computational complexity over one train epoch is then proportional to $O((C + \frac{V}{C}) \times H)$. The minimum computation is obtained when C equals $\sqrt{V}$, a computational reduction of $\sqrt{V}$ could be obtained compared to F-RNNLMs in Equation 4.20 when output vocabulary is significantly larger than hidden layer size H (i.e. $V >> H$). Similar computational reductions on error backpropagation and update in the output layer are also achieved. This structure gives a large speedup on both training and inference stages for RNNLMs [154].

When variance regularisation is used for training criterion, the computational load in the train stage is the same for each sample. In the test time, the computational complexity for each sample in the output layer becomes H, which is the hidden layer size. For noise contrastive estimation, the computational complexity during the train stage in the output layer is proportional to,

$$H \times k \tag{4.23}$$

where k is the number of noise sample. The test computation is the same as variance regularisation, which is H. Table 4.1 gives the computational complexities of RNNLM using different output layer structures and training criteria for one sample in the forward process. The computational loads in the hidden layer are not shown in the table since they are the same regardless of the output layer structure and training criteria.

4.4 Implementation

In order to facilitate parallelisation on GPU, a bunch mode can be applied to neural networks. This technique has previously been used for feedforward NNLMs [195, 196]. A fixed number (i.e. bunch) of n-grams could be collected from the training data. They are propagated through the network and accumulated the gradients. The update of weight parameters is based on the gradient over the whole bunch. The fixed number of training samples, or bunch size, is normally chosen between 2 and 256 for feedforward NNLMs [196]. This form of bunch based training facilitates the matrix operation, thus making it more suitable for the implementation of GPU and giving significant speedup in train speed. However, there is scarce work on applying bunch mode for the training of RNNLMs. In this thesis, a novel sentence splice method is proposed to arrange the data structure, which is more suitable for bunch model based training. The strong parallel power of GPU is also explored and significant acceleration can be obtained.

4.4.1 RNNLM training word by word

In terms of implementation, back propagation through time (BPTT) [236] can operate on sentence level or word level. In the former case, after seeing the complete sentence, the error computed for each word is back propagated to the beginning of the sentence. This is the implementation in Tensorflow [1] and RWTH LM [218]. An alternative way of implementating BPTT for RNNLMs is to update word by word. RNNLM toolkit [155] provides an example of this implementation. Instead of updating based on complete sentences, in this fashion, update is word by word. For each word, forward is carried out first, then followed by back propagating error through time and updating model parameters. These two types of implementation are described in this section.

RNNLM training sentence by sentence: The blue flow in Figure 4.4 illustrates the forwarding process associated with a sentence, where, $h^{(I)}$ is the initial recurrent vector. The recurrent vectors $h_i(i = 0,1,...,8)$ are computed and stored during the process. These recurrent vectors will be used in the following BPTT and model update. The updated model parameters are then used for the next sentence.

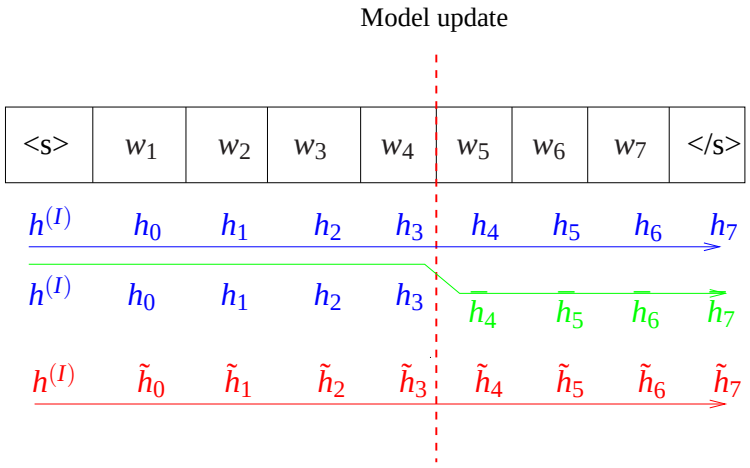


Fig. 4.4 *RNNLM training in one sentence. The blue line shows the process of sentence by sentence update and the recurrent vectors are computed by the same model without update until seeing the complete sentence; the green and read line are for word by word update. The green line shows the update of word w_5 using approximate recurrent vectors generated by old model parameters. The red line shows the correct way for word by word update, all history recurrent vectors are re-computed after each update*

RNNLM training word by word : consider now updating model parameters in word by word basis. Figure 4.4 illustrates this for word w_5. After processing word w_4, the recurrent vector $\bar{h}_4$ is computed. In order to use the correct statistics for BPTT and update, the previous recurrent vectors should be re-calculated from the sentence start. These recurrent vectors are indicated as $\tilde{h}_i(i = 0,1,2,3)$ and this process is shown as red flow in Figure 4.4.

However, the re-computation of all previous recurrent vectors from sentence start is computationally expensive for the update of each word. Hence, an approximation is introduced, where the recurrent vectors $h_i(i = 0,1,2,3)$ are obtained using the old model parameters $\mathcal{M}$

to yield the approximate history recurrent vectors $\tilde{h}_i(i = 0,1,2,3)$. Although, they are not the correct recurrent vectors due to the mismatch between h_i and $\tilde{h}_i$, this approximation was empirically found not to hurt performance [150]. This process is illustrated as green flow in Figure 4.4.

In this thesis, we adopted RNNLM training word by word. This implementation facilitates parallelisation and allows more sentences to be processed in parallel during training as will be shown in the Section 4.4.4.

4.4.2 Conventional Bunch Mode RNNLM Training

It is easy to apply bunch mode training to feedforward NNLM since the *n*-grams can be collected before training. However, it is not so straightforward for RNNLMs training to use bunch mode training. As RNNLMs use a vector to represent the complete history, the predicted probability of the current word depends on the whole history information. Hence, each sentence has to be processed in order from the begining to end. Instead of operating at the *n*-gram level, a sentence level bunch is used [207, 217]. This form of parallelisation requires each sentence to be regarded as independent in RNNLM training by re-initialising the recurrent hidden history vector at the start of every sentence.

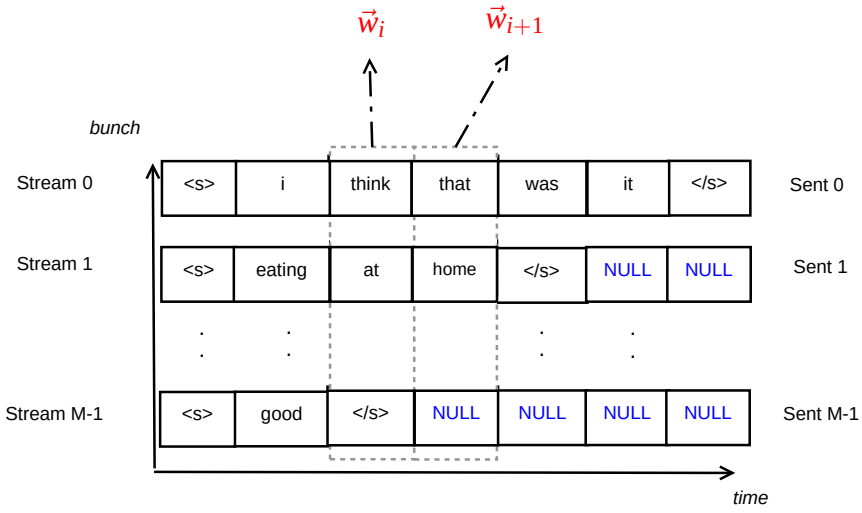


Fig. 4.5 *An example of bunched RNNLM training without sentence splicing. NULL tokens are added in the end of sentences to get the same sentence length for all sentences*

The basic idea of bunch mode training is shown in Figure 4.5. Assuming the bunch size is M and the whole corpus contains S sentences, a total of M sentences are aligned from left to right. During parallelisation, a regular structured input matrix is formed. The element at the j^{th} row and i^{th} column in the input matrix, associated with time $i+1$ and an output word $w_{i+1}^{(j)}$, represents a vector $[w_i^{(j)}, v_{i-1}^{(j)}]^\top$, where $w_t^{(j)}$ and $v_{i-1}^{(j)}$ are the 1-of-k vector encoding of the i^{th} word of the j^{th} sentence in the bunch, and the corresponding recurrent history vector at word $w_i^{(j)}$ respectively.

Two issues arise when directly using the above sentence bunch mode training. First, the variation of sentence length in the training data requires the number of columns of the input matrix to be the maximum sentence length in the training corpus. To handle this issue, NULL "words" are then appended to the end of other shorter sentences in the bunch, as shown in Figure 4.5. These redundant NULL words are ignored during BPTT. As the ratio between the maximum and average sentence length of the training data increases, its potential speed up from parallelisation is descreased. Second, the conventional sentence bunch mode training also interacts with the use of class based RNNLMs [154, 217]. As words aligned at the same position across different sentences can belong to different classes, the associated output layer submatrices of irregular sizes will be used at the same time instance. This can also result in inefficiency during training.

4.4.3 Bunch Mode RNNLM Training with Sentence Splicing

To improve efficiency, bunch mode based on spliced sentences is proposed in this thesis. Instead of a single sentence, each stream in the bunch now contains a sequence of concatenated sentences, as illustrated in Figure 4.6. Sentences in the training corpus now can be concatenated into streams that are more comparable in length. Individual sentence boundaries within each stream are marked in order to appropriately reset the recurrent history vector as required. As the streams are more comparable in length, the insertion of NULL tokens at the stream end is minimised. This approach can thus significantly reduce the synchronisation overhead and improve the efficiency in parallelisation.

The bunch mode training of C-RNNLMs requires the use of different submatrices as words within the same bunch maybe from different classes, which complicates implementation. The non-class based, full output layer RNNLMs (F-RNNLMs) introduced in Chapter 4.1.1 are chosen to avoid this issue. F-RNNLMs use the entire output layer both in training and LM probability calculation, therefore allowing the speed improvements from parallelisation techniques to be fully exploited.

It is also worth noting that, we can not apply sentence splice technique for the sentence by sentence update. Otherwise, the sentence boundary will appear in the middle of spliced sequence randomly as shown in Figure 4.6. Another solution to improve parallelisation efficiency for the sentence by sentence RNNLM training is to choose multiple sentence with similar sentence length. However, we found this may affect the performance by clustering sentences with similar length.

4.4.4 Analysis of the Efficiency of Parallelisation

As discussed in Section 4.4.1, the RNNLMs could be updated either a sentence by sentence, or word by word basis. Both of these two implementations can be parallelised by aligning multiple sentences. In this section, we will compare the efficiency of these two implementations. For simplicity, we assume that M sentences are parallelised for computation and all sentences have the same sentence length N as shown in Figure 4.7

In our implementation, model parameters are updated on the word by word basis. Therefor N updates are required to process $M \times N$ words shown in Figure 4.7. Each update collects

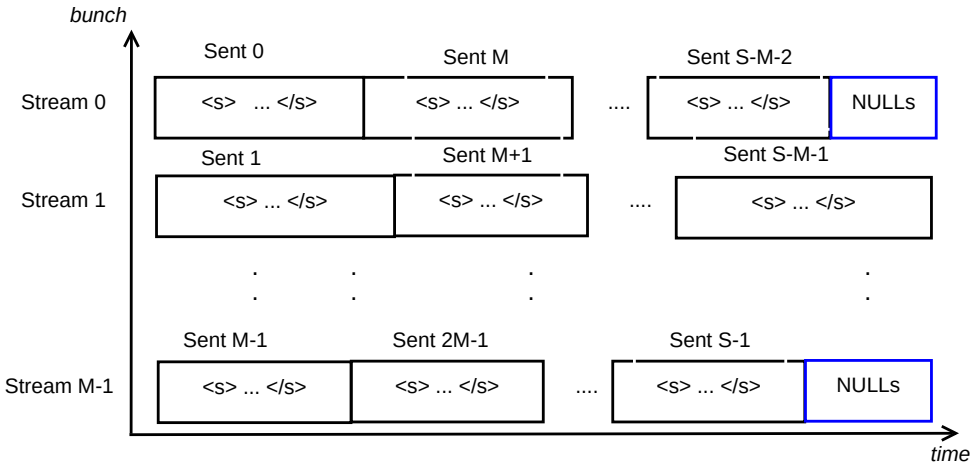


Fig. 4.6 *An example of bunched RNNLM training with sentence splicing. All sentences from the training corpus are concatenated into M long streams and NULL tokens are only needed at the end of the training corpus.*

gradients of M words from the same column as shown in Equation 4.24.

$$\bigtriangledown = \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i}^{M} \ln P(w_j^{(i)} | h_j^{(i)}; \boldsymbol{\theta}) \tag{4.24}$$

For the sentence by sentence update, only 1 update is required and the gradients are collected from M sentences as shown in Equation 4.25. However, the computation is only parallelised over M sentences as words in a sentence still need to be processed from left to right.

$$\bigtriangledown = \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i}^{M} \ln P(\mathcal{W}^{(i)}; \boldsymbol{\theta}) = \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{j}^{N} \sum_{i}^{M} \ln P(w_j^{(i)} | h_j^{(i))}; \boldsymbol{\theta}) \tag{4.25}$$

When applying SGD for RNNLM training, a proper minibatch size needs to be set. A small minibatch size may cause slow training and take longer time to train; a large minibatch may hurt convergence and degrade performance. For example, 128 was empirically found optimal minibatch size for a range of tasks [40, 239, 112]. During word by word based update, we can set M to be 128 and parallel 128 sentences for training. During sentence by sentence based updating, if we assume that the sentence length N is 10 and still use $M = 128$. A total of 1280 words will be used for each update. This will result in a bad convergence. A comparable value for M in the sentence by sentence update mode should be about 13. In this case, only 13 sentences will be parallelised during RNNLM training. Hence, we can see that our implementation is more beneficial for parallelisation.

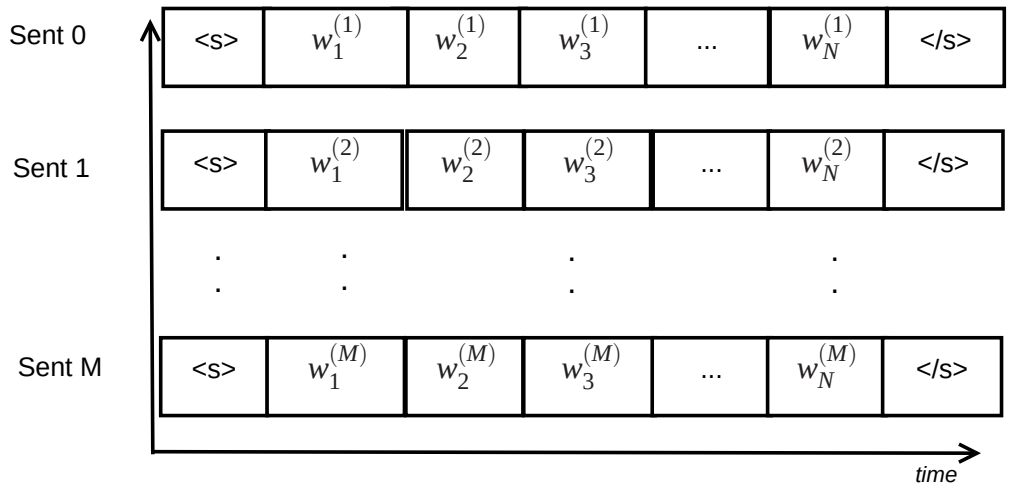


Fig. 4.7 *An example of M sentences with the same sentence length N.*

4.4.5 Efficient Softmax Calculation and Parameter Tuning

To improve efficiency, graphics processing units (GPUs), which have been previously employed to train deep neural network based acoustic models in speech recognition [37, 199], are used to train RNNLMs. CUBLAS from CUDA 5.0, the basic linear algebra subprograms (BLAS) library optimised for Nvidia GPUs, is used for fast matrix operation. As discussed in Chapters 4.1.2, when a large number of output layer nodes is used, the softmax computation during gradient calculation is very expensive. To handle this problem, a fast GPU implementation of the softmax function is used. Instead of summing the sufficient statistics sequentially over all output layer nodes, they are processed in one block. Shared memory is also used to facilitate rapid address access time. An array in each block with a fixed length (1024 used in this work) is allocated in the shared memory. Partial accumulates are stored in the array elements. A binary tree structured summation performed over the array reduces the execution time from M to $\log M$, for example, from 1024 down to 10 parallelised GPU cycles.

In order to obtain a fast and stable convergence during RNNLM training, the appropriate setting and scheduling of the learning rate parameter are necessary. For the F-RNNLMs trained with bunch mode, the initial learning rate is empirically adjusted in proportion to the underlying bunch size. When the bunch size is set to 1 and no form of parallelisation is used, the initial learning rate is set to 0.1, in common with the default setting used in the RNNLM toolkit [155]. The initial learning rate settings used for various other bunch sizes are also given in Table 4.2. When the bunch size increases to 128, the initial learning rate is set to 0.0078 per sample.

Table 4.2 The empirical value for initial learning rate per sample with different bunch size for RNNLM training.

bunch size	1	8	32	64	128	256
learning rate	0.1	0.0375	0.025	0.0156	0.0156	0.0078

4.5 Pipelined Training of RNNLMs

A practical way to further speedup training is using more machines (CPUs or GPUs) for parallel computation. The parallel training of neural network can be split into two categories: model parallelism and data parallelism [200]. The difference lies in whether the model or data is split across multiple machines or cores. Pipelined training is a type of model parallelism. It was first proposed to speedup the training of deep neural network (DNN) based acoustic models in [37]. Layers of the network are distributed across different GPUs. Operations on these layers such as the forward pass and error back propagation are executed on their own GPUs. It allows each GPU to proceed independently and simultaneously. The communication between different layers is performed after each parameter update step. In this thesis the idea of pipelined training is to be applied in the training of RNNLMs.

An example of RNNLM with one hidden layer and associated data flow of pipelined training is shown in Figure 4.8. The hidden layer (denoted by Weight 0) and output layer weight (denoted by Weight 1) matrices are kept in two GPUs (denoted by GPU 0 and GPU 1). For the first bunch in each epoch, the input is forwarded to the hidden layer and the output of hidden layer is copied from GPU 0 to GPU 1. For the 2nd bunch, the input is forwarded to hidden layer in GPU 0. Simultaneously, GPU 1 forwards the previous bunch obtained from hidden layer to the output layer. This is followed in sequence by error back propagation, parameter update, and the communication between GPUs in the form of a copying operation. For the following bunches, GPU 0 updates the model parameters using the corresponding error signal and input with BPTT, before forwarding the new input data for the next bunch. GPU 1 performs successively a forward pass, error back propagation and parameter update again.

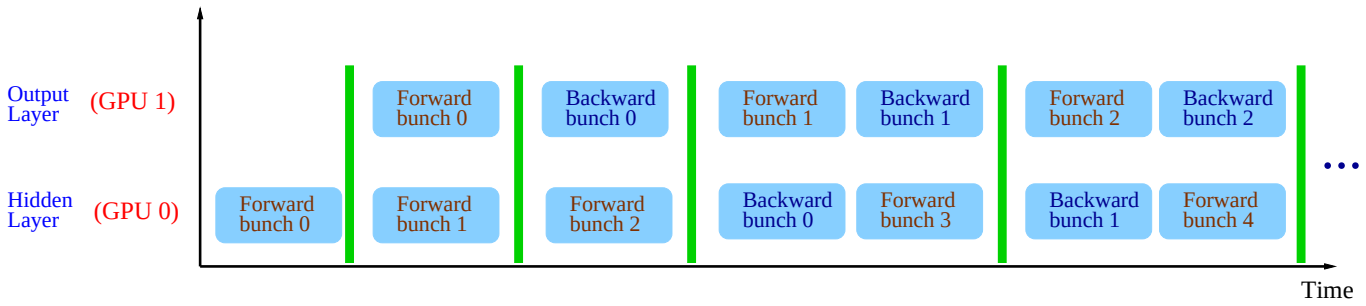


Fig. 4.8 *An example of data flow in pipelined RNNLM training using 2 GPUs*

4.6 Experiments

In this section, the performance of the proposed techniques to improve RNNLM training and inference efficiency are evaluated on two tasks: a HTK-based large vocabulary speech recognition system developed for English conversational telephone speech (CTS) used in the 2004 DARPA EARS evaluation [64]; and Google's one billion word benchmark corpus [32] for language modelling.

4.6.1 CTS-English ASR Experiment

In this section, RNNLMs are evaluated on the CU-HTK LVCSR system for conversational telephone speech (CTS) used in the 2004 DARPA EARS evaluation. The MPE [178] acoustic models were trained on approximately 2000 hours of Fisher conversational speech released by the LDC. A 59k recognition word list was used in decoding. The system uses a multi-pass recognition framework. The initial lattice generation used adapted gender dependent cross-word triphone MPE acoustic models with HLDA projected, conversational side level normalised PLP features, and unsupervised MLLR [68] speaker adaptation. A pruned interpolated 3-gram LM was used for lattice generation and followed by lattice rescoring using an unpruned 4-gram LM. A more detailed description of the baseline system can be found in [65]. It is worth mentioning that the system was built in CUED for evaluation in about 10 years before, which cannot reflect the performance of state-of-the-art acoustic model. For the performance of RNNLM on advanced acoustic models, the readers are referred to Chapter 8, where meeting data is used to construct state-of-the-art ASR system. The 3 hour **dev04** data, which includes 72 Fisher conversations and contains on average 10.8 words per segment, was used as a test set. For results presented in this chapter, matched pairs sentence-segment word error (MAPSSWE) based statistical significance test was performed at a significance level $\alpha = 0.05$ to ensure the improvement is statistically significant. The baseline 4-gram LM was trained using a total of 545 million words from two text sources: the LDC Fisher acoustic transcriptions, **Fisher**, of 20 million words (weight 0.75), and the University Washington conversational web data [24], **UWWeb**, of 525 million words (weight 0.25). The **Fisher** data was used to train various RNNLMs. A 38k word input layer vocabulary and 20k word output layer shortlist were used. RNNLMs were interpolated with the baseline 4-gram LM using a fixed weight 0.5. This baseline LM gave a WER of 16.7% on **dev04** measured using lattice rescoring.

The baseline class based RNNLMs were trained on CPU with the modified RNNLM toolkit [155] compiled with g++ [5]. The number of BPTT steps was set as 5. A computer with dual Intel Xeon E5-2670 2.6GHz processors with a total of 16 physical cores was used for CPU-based training. The number of classes was fixed as 200. The number of hidden layer nodes was varied from 100 to 800. 12 epochs were required for RNNLMs training to get convergence in this task. The 100-best hypotheses extracted from the baseline 4-

[5] A speedup of 1.7 times for CPU based training could be obtained by the Intel MKL CUBLAS implementation with multi-threading (compiled with icc version 14.0.2) over the baseline RNNLM toolkit for C-RNNLMs with 512 hidden layer nodes and 200 classes.

gram LM lattices were rescored for performance evaluation. The perplexity and error rates of various RNNLMs are shown in Table 4.3. The C-RNNLM with 512 hidden layer nodes gave the lowest WER of 15.3% and serves as the baseline C-RNNLM in all the experiments.

Table 4.3 Training speed, perplexity and WER results of CPU trained C-RNNLMs on the CTS task with varying hidden nodes with cross entropy based training.

hidden nodes	speed (w/s)	train time (hours)	**dev04**	
			PPL	WER
4glm (baseline)			51.8	16.72
100	7.6k	9.8	50.7	16.13
200	2.1k	35.6	48.6	15.82
512	0.37k	202.1	46.5	**15.32**
800	0.11k	679.9	45.8	15.40

Experiment on bunch mode training on GPU

The next experiment was to examine the efficiency of bunch mode GPU based RNNLM training with sentence splicing. The Nvidia GeForce GTX TITAN GPU was used to train various F-RNNLMs. The spliced sentence bunch mode parallelisation algorithm and its GPU implementation described in Chapters 4.4 were used. A range of different bunch size settings from 8 to 256 were used. Consistent with the above C-RNNLM baseline, all F-RNNLMs have 512 hidden layer nodes and a comparable number of weight parameters. Their performance measured in terms of training speed, perplexity and WER are shown in the 2nd Section of Table 4.4. To illustrate the performance differences among RNNLMs, WERs are shown with accuracy of 2 decimal places. The performance of the baseline C-RNNLM with 512 hidden nodes (shown in the 4th line in Table 4.3) is again shown in the 1st line of Table 4.4. Setting the bunch size to 8, a 4 times speed up is achieved. Improvements in perplexity and WER over the C-RNNLM baseline are also obtained. Further improvements in training speed can be consistently achieved by increasing the bunch size to 128 without performance degradation. The best performance in terms of training speed and WER was obtained by using a bunch size of 128. This gives a 27 times speed up and a 0.1% absolute reduction in WER over the C-RNNLM baseline.

Examining the breakdown of the training time suggests the output and hidden layers account for the majority of computation during BPTT (44.8% and 39.4% respectively), due to the heavy matrix multiplications required. The remaining computation is shared by other operations such as resetting F-RNNLM hidden vectors at the sentence start, and data transfer between the CPU and GPU. This breakdown of training time suggests that further speedup is possible via pipelined training by allocating the computation of the hidden layer and output layer into different GPUs, as shown latter in Table 4.10. A further speed up is possible, for example, by increasing the bunch size to 256. However, the convergence becomes unstable and leads to performance degradation.

Table 4.4 Training speed, perplexity and WER results of GPU trained F-RNNLMs on **dev04** with varying bunch sizes and a fixed hidden layer size of 500.

model type	bunch size	#parameter	speed (w/s)	time (hours)	**dev04**	
					PPL	WER
C-RNN	-	26.9M	0.37k	202.1	46.5	15.32
F-RNN	8	26.8M	1.4k	53.4	45.7	15.22
	32		4.6k	16.3	45.6	15.25
	64		7.6k	9.8	45.7	15.16
	128		10.1k	7.4	46.3	15.22
	256		12.9k	5.7	46.5	15.38

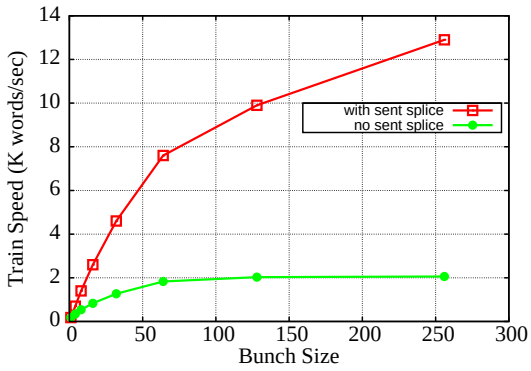


Fig. 4.9 *F-RNNLM training speed with and without sentence splicing on the CTS task. The red line is the train speed with spliced sentence bunch training and the green line is without sentence bunch but only aligning multiple sentences.*

An analysis of number of NULL tokens with and without sentence splicing was carried out. With bunch size 128, by simply aligning the sentences, 72M NULL tokens are appended at the end of sentence in one epoch. In contrast, only 60K NULL tokens are necessary at the end of each epoch when sentence splicing is adopted. It could been seen that the number of NULL token can be reduced dramatically. As a major contribution factor to the above speed improvements,the importance of using sentence splicing in bunch mode based GPU implementation is shown in Figure 4.9, where a contrast in speed with and without sentence splicing is drawn. When using the conventional bunch model training with no sentence splicing as shown in Figure 4.5, only limited speed improvements were obtained by increasing the bunch size. This is due to the large number of inserted NULL tokens and the resulting inefficiency, as discussed in Section 4.4. These results suggest that the proposed sentence splicing technique is important for improving the efficiency of bunch mode RNNLM training.

The spliced sentence bunch based parallelisation can also be used for RNNLM performance evaluation on GPU. Table 4.5 shows the speed information measured for N-best rescoring using the baseline C-RNNLM and the F-RNNLM of Table 4.4. As expected, it is very expensive to use F-RNNLM on CPU, which is discussed in the following section.

C-RNNLMs can improve the speed by 43 times. A further speedup of 9 times over the CPU C-RNNLM baseline was obtained using the bunch mode (bunch size 512) parallelised F-RNNLM.

Table 4.5 Evaluation speed of C-RNNLMs (class based RNNLMs) and F-RNNLMs (full output layer RNNLMs) for N-Best scoring on **dev04**. The F-RNNLM is trained with bunch size 128.

model type	device	bunch size	test speed (words/sec)	WER
C-RNN	CPU	N/A	5.9k	15.32
F-RNN			0.14k	15.22
F-RNN	GPU	1	1.1k	15.22
		64	41.3k	
		512	56.3k	

Experiment on variance regularisation

In this section, the performance of F-RNNLMs trained with variance regularisation are evaluated. These experimental results with various settings of the regularisation constant γ in equation (4.9) are shown in Table 4.6. The word error rates with $Z(h)$ in the table are the WER scores measured using the conventional normalised RNNLM probabilities computed using Equation 4.6. WERs with Z in the last column are obtained using the more efficiently approximated RNNLM probabilities given in Equation 4.12. The first row of Table 4.6 shows the results without variance regularisation by setting γ to 0, the same to the conventional CE based training. As expected, the WER was increased from 15.22% (conventional fully normalised F-RNNLMs) to 16.24% without performing the normalisation. This confirms that the normalisation term computation for the softmax function is crucial for using cross entropy (CE) trained RNNLMs in decoding.

When the variance regularisation term is applied in F-RNNLM training, there is only a small difference in terms of WER between using the accurate normalisation term $Z(h)$ or approximate normalisation term Z. As expected, when the setting of γ is the increased, the variance of the log normalisation term is decreasing. When γ is set as 0.4, it gave a WER of 15.28%, insignificant to the WER of the baseline CE trained F-RNNLM (1st line in Table 4.6 and also 5th line in Table 4.4). At the same time, significant improvements in evaluation speed were also obtained. This is shown in Table 4.7. The CPU based F-RNNLM evaluation speed was increased by a factor of 56 over the CE trained F-RNNLM baseline using variance regularisation, while retaining the same training speed.

Experiment on NCE training

The next experiment was using NCE for RNNLM training, where accelerations in both training and decoding are expected since the computation in the output layer can be reduced

Table 4.6 Perplexity and WER performance of F-RNNLMs trained with variance regularisation on **dev04**. The mean and variance of log normalisation term were computed over the validation data. The two columns under WER (Z(h) and Z) denote word error rates using normalised or approximated RNNLM probabilities computed using Equations 4.6 and 4.12.

γ	log-norm		PPL	WER	
	mean	var		Z(h)	$\overline{\ln Z}$
0.0	15.4	1.67	46.3	15.22	16.24
0.1	14.2	0.12	46.5	15.21	15.34
0.2	13.9	0.08	46.6	15.33	15.35
0.3	14.0	0.06	46.5	15.40	15.30
0.4	14.2	0.05	46.6	15.29	15.28
0.5	14.4	0.04	46.5	15.40	15.42

Table 4.7 Training and evaluation speed of F-RNNLMs trained with variance regularisation on the CTS task. C-RNNLMs were trained on CPU and F-RNNLMs on GPU. Both were evaluated on CPU.

model type	train crit	train speed(w/s)	train time (hours)	test speed(w/s)
C-RNN	CE	0.37k	202.1	5.9k
F-RNN	CE	10.1k	7.4	0.14k
	VR	10.1k	7.4	7.9k

significantly. As discussed in Chapters 4.2.3, an important attribute of NCE training is that the variance of the RNNLM output layer normalisation term Z can be implicitly constrained to during parameter estimation. This effect is illustrated in Figure 4.10 on the log scale over a total of 12 epochs on the validation data set. The variance of the normalisation term is slightly increased from 0.035 to 0.06 in the first 4 epochs, then gradually reduced to 0.043 at the last epoch.

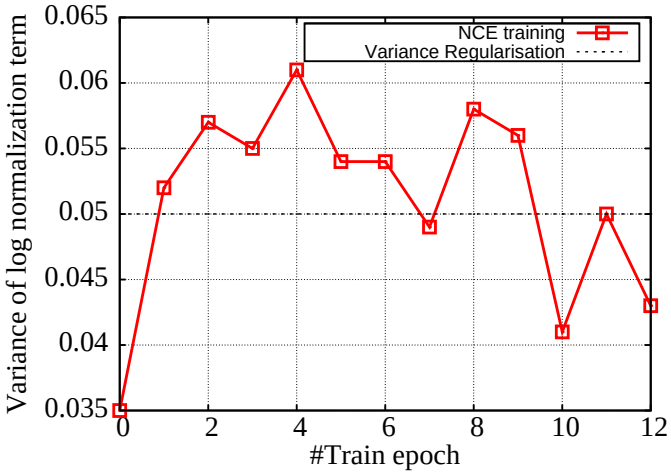


Fig. 4.10 *Variance of the output layer log normalisation term* $\overline{\ln Z}$ *on the validation data on CTS task at different epochs during NCE based RNNLM training.*

The WER and PPL results of NCE trained RNNLM are given in Table 4.8. 12 epochs were required for both the conventional CE and NCE based training to converge. As discussed in Chapter 4.2.3, the log normalisation term $\overline{\ln Z}$ in Equation 4.19 was fixed as 9. The perplexity scores in Table 4.8 were obtained by explicitly computing the output layer normalisation term. During N-best rescoring, normalised RNNLM probabilities were used for the CE trained RNNLM baseline, while unnormalised probabilities were used for the NCE trained RNNLM. As is shown in Table 4.8, the NCE trained RNNLM gave slightly worse performance than the CE trained baseline. At the same time, the train speed was doubled. This is expected as the time consumed on output layer is approximately half of the total training time required for conventional CE training.

Similarly a large testing time speedup of 56 times over the CE trained RNNLM on CPUs was also obtained, as is shown in Table 4.8. This improvement is comparable to the speedup obtained using variance regularisation based RNNLM training previously shown in Table 4.7. As the computation of the normalisation term is no longer necessary for NCE trained RNNLMs, the computational cost incurred at the output layer can be significantly reduced.

As expected, the NCE training speed is also largely invariant to the size of the output layer, and thus improves the scalability of RNNLM training when a very large output vocabulary is used. This highly useful feature is clearly shown in Table 4.9, where CE training speed decreases rapidly when the output layer size increases. In contrast, the NCE training speed remains constant against different output layer vocabulary sizes.

Table 4.8 Perplexity and WER performance, training and evaluation speed of NCE trained F-RNNLMs on the CTS task. C-RNNLMs trained on CPU and F-RNNLMs on GPU. Both evaluated on CPU.

model type	train crit	train speed(w/s)	train time(hr)	test speed(w/s)	**dev04**	
					PPL	WER
C-RNN	CE	0.37k	202.1	5.9K	46.5	15.32
F-RNN	CE	10.1k	7.4	0.14k	46.3	15.22
	VR	10.1k	7.4	7.9k	46.6	15.28
	NCE	19.7k	3.8	7.9k	46.8	15.37

Table 4.9 The training speeds (w/s) using cross entropy (CE) and noise contrastive estimation (NCE) with different output layer size for F-RNNLMs on CTS task

train crit	#output layer nodes		
	20k	35k	30k
CE	10.1k	9.1k	8.0k
NCE	19.7k		

Experiment on Dual GPU pipelined training

In this section, the performance of a dual GPU based pipelined F-RNNLM training algorithm is evaluated. In the previous experiments, a single Nvidia GeForce GTX TITAN GPU (designed for a workstation) was used. For the pipelined training experiments, two slightly slower NVidia Tesla K20m GPUs housed in the same server were used. Table 4.10 shows the training speed, perplexity and WER results of pipelined CE training for F-RNNLMs. As is shown in the table, pipelined training gave a speedup of a factor of 1.6 times and performance comparable to a single GPU based training.

Table 4.10 Training speed, perplexity and WER performance of F-RNNLMs on **dev04** using pipelined CE training on CTS task.

model type	GPU	train speed(w/s)	**dev04**	
			PPL	WER
C-RNN	-	0.37k	46.5	15.32
F-RNN	1xTITAN	10.1k	46.3	15.22
	1xK20m	6.9k	46.3	15.22
	2xK20m	11.0k	46.3	15.23

4.6.2 Google's One Billion Word Experiment

A new benchmark corpus was released by Google for measuring performance of statistical language models [32]. Two categories of text normalisation are provided. One is for ma-

chine translation (StatMT) and the other is for ASR provided by Cantab Research [6]. The latter was used to further evaluate the performance and scalability of NCE based RNNLM training in this experiment for training RNNLMs on large corpus. The former will be used to build language model in the next experiment. A total of 800 million words were used in LM training. A test set of 160k words (obtained from the first split from held-out data) was used for perplexity evaluation. A modified KN smoothed 5-gram LM was trained using the SRILM toolkit [213] with zero cut-offs and no pruning. In order to reduce the computational cost in training, an input layer vocabulary of 60k most frequent words and a 20k word output layer shortlist were used. RNNLMs with 1024 hidden layer nodes were either CE or NCE trained on a GPU using a bunch size of 128. The other training configurations were the same as the experiments presented in Chapter 4.6.1. A total of 10 epochs were required to reach convergence for both CE and NCE based training. The perplexity performance of these two RNNLMs are shown in Table 4.11. Consistent with the trend found in Table 4.8, the CE and NCE trained RNNLMs gave comparable perplexity when interpolated with the 5-gram LM. A large perplexity reduction of 21% relative over the 5-gram LM was obtained.

Table 4.11 Perplexity performance of RNNLMs on Google's one billion (for ASR (provided by Cantab Research) corpus) word corpus.

LM	train crit	train speed(w/s)	PPL	
				+5glm
5glm	-	-	83.7	-
F-RNN	CE	6.7k	104.4	65.8
	NCE	11.3k	107.3	66.0

In order to further investigate the scalability of NCE based RNNLM training, an additional set of experiments comparable to those presented in Table 4.11 were conducted using a much larger, full output layer vocabulary of 793k words as used in previous research [52]. It is worth mentioning that the corpus used in this experiment is for machine translation, which is different to the corpus used for ASR provided by Cantab Research in previous experiments. Due to the large size of such output layer vocabulary, the speed of standard cross entropy training of a full output layer based RNNLM is as slow as 200 words per second. It is therefore computationally infeasible in practice to train such a baseline RNNLM. The training speed and perplexity score of a NCE trained RNNLM with a 793k vocabulary are presented in Table 4.12, together with the baseline 5-gram LM's perplexity performance. A total of 1024 hidden nodes and a bunch size of 128 were used. The stand alone NCE trained RNNLM gave a perplexity score of 77.3. This was further reduced to 52.6 after an interpolation with the 5-gram LM[7]. Note that in order to ensure stable convergence during NCE training, an additional gradient clipping step was also applied. In combination with drawing noise samples over a much larger output layer, this additional operation led to only

[6] All sources are available in https://code.google.com/p/1-billion-word-language-modeling-benchmark/. The machine translation normalised version of this data was previously used in [32] for RNNLM training.

[7] The log file for perplexity computation is also available in http://mi.eng.cam.ac.uk/projects/cued-rnnlm/ppl.h1024.log

a moderate decrease in the training speed from 11.3k to 6.5k words per second, when compared with the NCE trained 20k vocabulary RNNLM in Table 4.11. The relative increase in training time is much lower than that of the output layer vocabulary size, by approximately 40 times.

Table 4.12 Perplexity performance of RNNLMs on Google's one billion word corpus using 793K vocabulary. The train is run on GPU and test is on CPU.

LM	train crit	train speed(w/s)	test speed(w/s)	PPL	
					+5glm
5glm	-	-	-	70.9	-
F-RNN	NCE	6.5k	37.1	77.3	52.6

4.7 Conclusions

This chapter studies the efficient training and inference of RNNLMs. A sentence splicing based bunch mode training is proposed to facilitate the bunch (i.e. minibatch) based training on GPUs. Compared to the popular RNNLM toolkit trained on CPU, a speedup of 27 times is obtained on a CTS task by training RNNLMs with bunch mode on GPU. Pipelined training using 2 GPUs is also investigated for the training of RNNLMs, which gives 1.6 times speedup. Besides the traditional cross entropy, variance regularisation (VR) and noise contrastive estimation (NCE) are also introduced into the training of RNNLMs to further improve the training and inference efficiency. RNNLMs trained by either VR and NCE give much faster inference speed than C-RNNLMs due to the use of unnormalised probability. Furthermore, NCE based training doubles the train speed. The experiment on Google one billion corpus indicates that the training scales well on a large amount of data.

Chapter 5

Efficient Lattice Rescoring of RNNLMs

The standard back-off *n*-gram language model (LM) is suitable for first-pass decoding and lattice rescoring due to its finite history character. The prediction of current word is only related to the previous *n*-1 words. During the search, histories from different paths can be combined according to their valid *n*-gram history, which reduces computation and memory demand significantly. However, for RNNLM, first-pass decoding and lattice rescoring become difficult for its long term character. There is no straightforward way to combine histories directly since the prediction of current word is associated with the complete history.

In order to address this issue, a range of techniques have been studied in recent years. Among these earlier works, a sampling based approach was used to generate text data from an RNNLM and a back-off *n*-gram LM is trained to approximate the RNNLM [59, 60]. A discrete quantisation of RNNLMs into a weighted finite state transducer (WFST) representation was proposed in [132]. An iterative lattice rescoring approach was first proposed and further investigated in [58]. However, these earlier schemes were unable to produce 1-best error rates comparable to the conventional N-best rescoring approach [59, 132] or generate a compact lattice representation of the hypothesis space that is suitable for downstream applications such as confusion network (CN) decoding [58, 107] or keyword search. Several latter works that were more successful in exploiting the lattice internal hypothesis ranking produced by an earlier decoding pass. This allows an approximate partial expansion of the underlying word graph to be performed during RNNLM rescoring [219, 220]. In this chapter, the lattice rescoring of RNNLM is studied and two algorithms for RNNLM lattice rescore are proposed [1]. This work enables efficient lattice rescoring using RNNLMs. Furthermore, compact lattices are generated after RNNLM rescoring and these lattices can be used by other applications such as confusion network decoding to get further word error rate improvement.

The chapter is organised as follow. Two widely used approaches, *n*-gram LM based approximation and N-best rescoring, for incorporation of RNNLM into speech recognition, are introduced in Chapter 5.1 and 5.2. The use of *n*-gram LM for history combination is reviewed in Chapter 5.3. Two proposed RNNLM lattice rescore methods are presented in

[1] This is a collaborative work with Dr Xunying Liu, who also co-supervised this work. The code for lattice expansion during RNNLM rescoring is mainly implemented by Dr Xunying Liu. The ideas are proposed during discussion.

Chapter 5.4, which are *n*-gram and history vector distance based clustering respectively. Experiments examine the proposed methods in Chapter 5.5 and the conclusion is given in Chapter 5.6.

5.1 *n*-gram LM approximation of RNNLM

RNNLM is difficult to directly apply to lattice rescoring due to its complete history character. One feasible approach is to build a language model to approximate the probability distribution of RNNLM and this language model is suitable for lattice rescoring, such as *n*-gram LM. The approximated language model can be optimised by maximising the following cross entropy objective function,

$$J^{\mathsf{CE}}(\boldsymbol{\theta}) = \sum_{(w,h)\in\mathcal{D}} P_{RNN}(w|h)\log\tilde{P}(w|h) \tag{5.1}$$

where $\mathcal{D}$ is the training data, $P_{RNN}(w|h)$ is the RNNLM probability and $\tilde{P}(w|h)$ is the probability of the approximated language model. This approximated language model can be any form of language model which is suitable for lattice rescoring. It is computationally intractable to optimise the parameter of $\tilde{P}(w|h)$ over all possible word and history (w,h). Sampling technique is used as approximation and the objective function shown in Equation 5.1 can be written as,

$$J^{\mathsf{CE}}(\boldsymbol{\theta}) = \sum_{(w,h)\in\mathcal{D}} \log\tilde{P}(w|h) \tag{5.2}$$

where $\mathcal{D}$ is the set of sentences randomly sampled from RNNLM $P_{RNN}(w|h)$. Equation 5.2 is the standard maximum likelihood (ML) objective function. In previous work [59], *n*-gram LM is chosen as the form of the approximated language model due to its finite history and easy incorporation for lattice rescoring.

In this method, RNNLM is used to randomly sample sentences. At the beginning of a sentence, the first input word is sentence start <s>, with a fixed and initialised history vector (e.g. all set to 0.1). A word is sampled according to the output layer distribution $P_{RNN}(w|h)$. The sampled word is used as input word for next sampling. This procedure can be applied repeatly until the sentence end symbol </s> is sampled. In this way, a large quantity of sentences can be generated from a well-trained RNNLM [60, 5]. An *n*-gram LM can be constructed on these sampled sentences which is then interpolated with the baseline *n*-gram LM. The resulting LM can be applied directly for first-pass decoding or lattice rescoring as an approximation to the original RNNLM. Previous research has shown that this approximated language model provides moderate gain over the baseline *n*-gram LM, but worse than the exact probability calculated using RNNLM [60].

5.2 N-best Rescoring of RNNLM

A nice feature of the approximation approach introduced above is that an *n*-gram language model with finite history can be constructed. However, a degradation of performance for speech recognition is observed compared to using exact probability from RNNLM. An alternative approach is to modify the lattice structure to make it suitable for RNNLM rescoring. Lattice is a direct acyclic graph (DAG) with compact representation of multiple hypotheses as shown in Figure 5.1. Each arc (i.e. word) in the lattice may have different histories. A parallel structure of the lattice can be constructed by unfolding all paths as shown in Figure 5.2. However, it is computationally intractable to list all possible paths in the lattice. N-best list only maintains the highest N paths from the lattice and allows the exact RNNLM probability P_{RNN} to be used. N-best rescoring is normally applied in speech recognition systems with an additional pass. The language model probability of these N-best hypotheses are substituted by the interpolated probabilities of the *n*-gram LMs and RNNLMs. The hypotheses are then reranked with the interpolated language model probability.

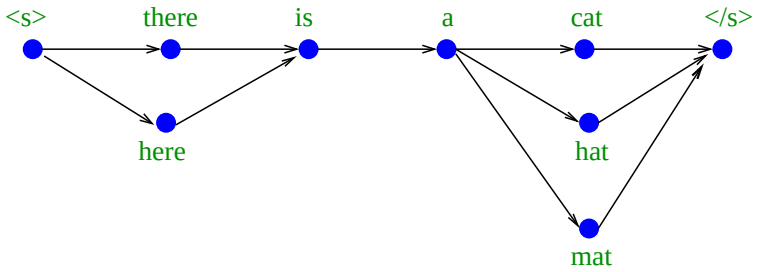


Fig. 5.1 *Example lattice for rescoring*

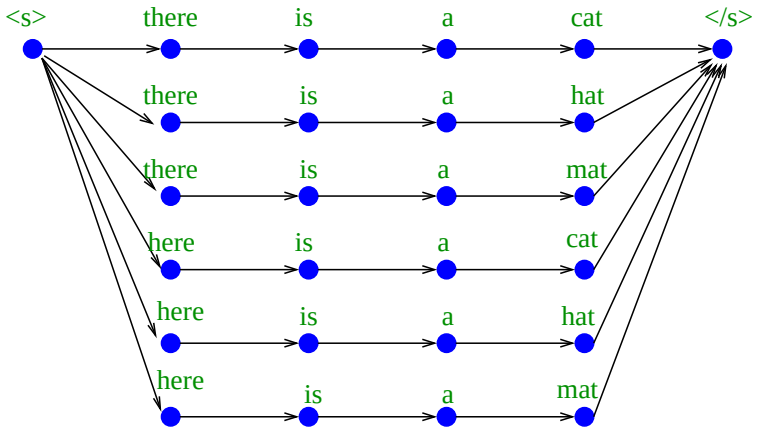


Fig. 5.2 *Example N-best list for rescoring*

N-best rescoring provides a straightforward way to incorporate RNNLM and promising performance can be obtained [154, 123]. However, only the top N-best sentences in the

lattice are possible decoding output and other possible hypotheses are ignored as the growth of N introduces a linear increase of computational load. A more computationally efficient way is to convert N-best into prefix for RNNLM rescoring [211]. The N-best list in Figure 5.2 can be converted to a prefix tree as shown in Figure 5.6, where the paths with the same history can be merged. However, it is still impractical to take into account all paths in the lattice given the vast amount of potential paths. Furthermore, N-best rescoring only updates the information for the N-best list, instead of the whole lattice. The output is 1-best hypothesis and it cannot generate lattice after N-best rescoring. The rich information of lattice is lost. Lattice is of practical value for various applications. Confusion network decoding [145] applied on lattice can obtain further recognition performance gain; the confidence score estimated from lattice is useful for for keyword search. Hence, RNNLM rescoring based on lattice is very useful for real applications.

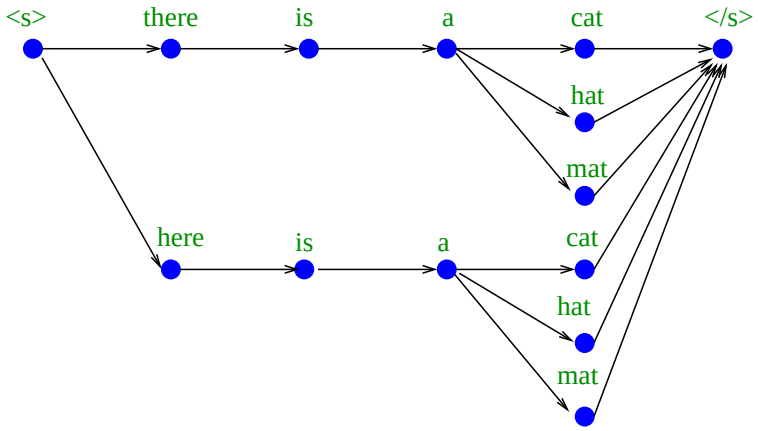


Fig. 5.3 *Example prefix tree for rescoring*

5.3 History Combination of *n*-gram LM

Before entering the discussion of RNNLM lattice rescoring, the history combination using *n*-gram LM based lattice rescoring is reviewed. In speech recognition, there is a large amount of possible paths during searching. In order to keep these candidates with a low computation and memory cost, the paths with the same *n*-gram LM history are merged as early as possible. As an example, when 2-gram LM is used, for the following two paths ending with words (w_{i-1}, w_i), their likelihoods are written as,

$$\begin{aligned} \mathcal{L}_1 &= p(\mathbf{o}_1^t | w_0^{i-2}, w_{i-1}, w_i) P(w_0^{i-2}, w_{i-1}) P(w_i | w_0^{i-2}, w_{i-1}) \\ \mathcal{L}_2 &= p(\mathbf{o}_1^t | \tilde{w}_0^{i-2}, w_{i-1}, w_i) P(\tilde{w}_0^{i-2}, w_{i-1}) P(w_i | \tilde{w}_0^{i-2}, w_{i-1}) \end{aligned} \tag{5.3}$$

The first item on the right of the above equation is acoustic model probability, the remaining two items are language model probability [2]. In a 2-gram LM, the language model probabil-

[2] language model scale and insertion penalty are ignored here

ities of the current word w_i are the same.

$$P(w_i|w_0^{i-2}, w_{i-1}) = P(w_i|\tilde{w}_0^{i-2}, w_{i-1}) \tag{5.4}$$

It is worth noting that the two history word sequences (w_0^{i-2}, w_{i-1}) and $(\tilde{w}_0^{i-2}, w_{i-1})$ are not required to have the same length. Moreover, for any words in the future, they also share the same language model probability. These two paths can be merged based on the discussion of Viterbi decoding in Chapter 2.4.1. The path with higher likelihood is kept and the other path is removed. This process is illustrated in Figure 5.4.

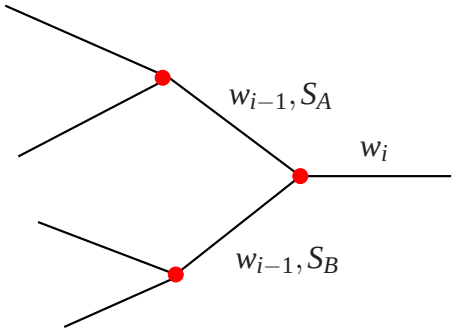


Fig. 5.4 *An example of combining two paths ending with (w_{i-1}, w_i) when 2-gram LM is used. S_A and S_B are the history states (e.g. acoustic model and language model likelihood, valid n-gram history) of these two paths.*

5.4 Lattice Rescoring of RNNLM

As stated above, it is impractical to compute the exact RNNLM probability for every possible path in a lattice, as it requires significant computational effort and results in prefix tree ultimately. Hence, some approximation is necessary to compress the search space.

Motivated by the path combination of *n*-gram LM discussed in Chapter 5.3, compact lattice is generated due to the *n*-gram assumption. This thought can also be applied on RNNLM to cluster similar histories so as to shrink the search space and reduce computation. The condition of combination of two paths in RNNLM can be written as,

$$P_{RNN}(w_i|w_0^{i-1}) \approx P_{RNN}(w_i|\tilde{w}_0^{i-1}) \tag{5.5}$$

Given two distinct histories w_0^{i-1} and $\tilde{w}_0^{i-1}$, if their probability distributions on current word w_i could be viewed equivalently, these two paths can be combined as an approximation.

The methods proposed in this thesis are inspired by two RNNLM assumptions. First, the history has a gradually diminishing effect on the predicted word probability in RNNLMs as the distance to the current word increases. This allows partial histories that are the same to share predicted word distribution. More precisely, if two finite histories share the same recent words, they are viewed as equivalent. It is thus possible to represent the infinite

history of RNNLMs using a truncated history with fixed and finite length, which is similar to *n*-gram LM. The 3-gram approximation for the RNNLM history combination can be shown as,

$$P_{RNN}(w_i|w_0^{i-3},w_{i-2},w_{i-1}) \approx P_{RNN}(w_i|\tilde{w}_0^{i-3},w_{i-2},w_{i-1}) \tag{5.6}$$

The probabilities of RNNLMs can be viewed as the same if the previous two words in the history are the same. Second, recalling to Equation 3.10, in a more general case, RNNLM probabilities can be represented by the previous word w_{i-1} and its history vector $\boldsymbol{v}_{i-2}$[3],

$$P_{RNN}(w_i|w_0^{i-1}) = P_{RNN}(w_i|w_{i-1},\boldsymbol{v}_{i-2}) \tag{5.7}$$

where $\boldsymbol{v}_{i-2}$ is the hidden history vector. Hence, it is also possible to explicitly use a hidden history vector distance based measure to determine the sharing of RNNLM probabilities. This method can be written as,

$$P_{RNN}(w_i|w_{i-1},\boldsymbol{v}_{i-2}) \approx P_{RNN}(w_i|\tilde{w}_{i-1},\tilde{\boldsymbol{v}}_{i-2}) \quad when \quad w_{i-1}=\tilde{w}_{i-1}, D(\boldsymbol{v}_{i-2},\tilde{\boldsymbol{v}}_{i-2})<\gamma \tag{5.8}$$

$D(\boldsymbol{v}_{i-2},\tilde{\boldsymbol{v}}_{i-2})$ denotes some distance measure to evaluate the similarity of these two hidden history vectors, which will be detailed below.

Motivated by these hypotheses, two efficient RNNLM lattice rescoring methods are proposed and investigated. The first uses an *n*-gram based clustering of history contexts [142, 104] and the second method exploits the distance measure between recurrent history vectors [142].

5.4.1 *n*-gram based History Clustering

This *n*-gram based history clustering is motivated by the assumption that the effect of distant history gradually vanishes, and only the most recent words have a large impact on predicting the next word. Two histories sharing the common $n-1$ previous words are viewed as equivalent. It is thus possible to merge these two paths into a single path for the following word predictions. Recalling the approximation of *n*-gram LM shown as Equation 2.22, a similar approximated RNNLM state for the complete history w_0^{i-1} can be written as,

$$\Phi_{RNN}(w_0^{i-1}) = w_{i-n+1}^{i-1} = <w_{i-1},\ldots,w_{i-n+1}> \tag{5.9}$$

The history state of RNNLM $\Phi_{RNN}(w_0^{i-1})$ during searching is decided by its previous $n-1$ words. When two paths are considered to be merged, their history states are compared. If their history states are equivalent, i.e. sharing the same previous $n-1$ words, these two paths are merged by keeping the path with higher likelihood and removing the other path with lower likelihood. This approximation reduces the computation and memory significantly. As the truncated history length $n-1$ increases, the approximated RNNLM probabilities are expected to be increasingly close to the true ones, while with a growth of computation. Figure 5.5 illustrates the history combination using 2-gram approximation. For the word

[3]The continuous history vector $\boldsymbol{v}_{i-1}$ can also be used to represent its complete history.

"a", it has two distinct history paths "<s> there is" and "<s> here is". The most recent word for both paths is "is". Hence these two paths could be combined and the history state of above path ("<s> these is") is kept due to its higher likelihood [4].

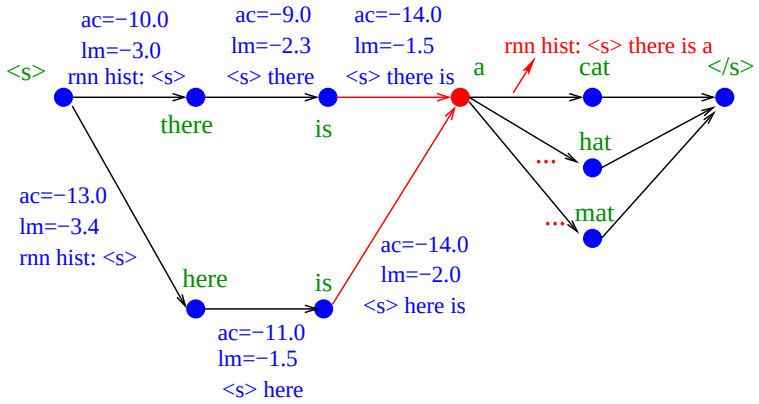


Fig. 5.5 *An example of n-gram based history clustering. For the node with word "a", its two history paths "<s> there is" and "<s> here is" has the same recent word "is". When 2-gram approximation is applied, these two paths can be merged and only the history path ("<s> there is") giving higher probability is kept and will be used for computing new history vector.*

The above history clustering algorithm in practice operates as a LM state cache that stores the RNNLM probabilities over the output vocabulary associated with distinct truncated *n*-gram histories derived from $\Phi_{NG}(\cdot)$. By default, if a particular truncated history state $\Phi_{NG}(w_0^{i-1})$ is not found in the cache, the full history w_0^{i-1} that subsumes the truncated context is used to create a new entry in the cache. As this algorithm uses similar information as conventional *n*-gram LM, it can be easily adapted and used by both beam search decoders [170, 168] where RNNLM probabilities can be computed on-the-fly by request and accessed via the cache. WFST [163] style lattice rescoring where a previously generated network can be used to build all possible shared RNNLM states into a WFST. In this thesis, only the lattice rescoring case is studied.

5.4.2 History Vector Distance based Clustering

The strong generalisation power of RNNLM is rooted in the continuous vector representation of word and history context. When clustering histories, it is also possible to make use of the similarity in their vector representation. The context state of an RNNLM is represented by an ordered pair that encodes the full, complete history $w_0^{i-1} =< w_{i-1}, \ldots, w_0 >$.

$$\Phi_{RNN}(w_0^{i-1}) \;=\; <w_{i-1}, \boldsymbol{v}_{i-2}> \tag{5.10}$$

[4] the language model scale and insertion penalty is ignored here.

The internal state of RNNLM here consists of two terms. The first item is the previous word w_{i-1}. The second item $\boldsymbol{v}_{i-2}$ gives a compact vector representation of history w_0^{i-2}. The history clustering can be based on this history representation directly. For two paths w_0^{i-1} and w_0^{j-1}, they will be clustered if the following constraints are satisfied:

$$w_{i-1} = w_{j-1} \quad \& \quad D(\boldsymbol{v}_{i-2}, \boldsymbol{v}_{j-2}) \leqslant \gamma \tag{5.11}$$

where γ is a threshold and can be viewed as a hidden history vector distance beam. $D(\cdot,\cdot)$ can be any distance measure such as Euclidean distance. It can be tuned flexibly to adjust the trade-off between precision and computation of this approximation.

Under this approximation, two paths with the common most recent word, and their distance of full history vector below a threshold γ are merged. For example, in the prefix tree shown in Figure 5.6, for the two nodes with word "a", its two histories "<s> there is" and "<s> here is" share the same previous word "is" and their hidden vectors are close enough. Hence, these two paths will be merged and the history vector of the above path which has a higher likelihood is kept.

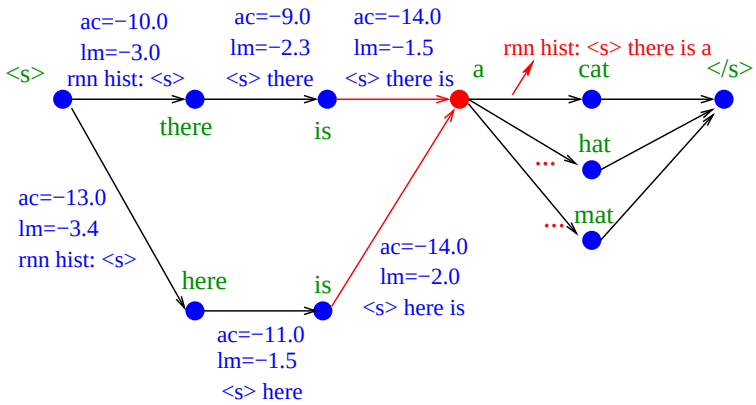


Fig. 5.6 *An example of history vector distance based clustering. For the node with word "a", its two history paths "<s> there is" and "<s> here is" can be represented by two history vectors in RNNLMs. These two paths can be merged as the Euclidean distance of these two vectors is smaller than a threshold. The history path ("<s> there is") giving higher probability is kept and will be used for computing new history vector.*

In similar form to the *n*-gram based clustering scheme introduced before, this history vector distance based clustering method is also implemented as a cache during lattice rescoring. This method can also be applied for beam search based decoder. However, due to the introduction of the distance beam γ, this technique is non-trivial to be directly used in generic WFST based decoding approaches.

There is a range of choices available for the distance measure $D(\boldsymbol{v}_{i-2}, \boldsymbol{v}_{j-2})$ over the two history vectors. The history vector is a vector with elements from the output of sigmoid function, which is bounded from 0 to 1. Here, the normalised Euclidean distance is used in

this thesis, which is given by,

$$D(\boldsymbol{v}_{i-2}, \boldsymbol{v}_{j-2}) = \frac{1}{d}\sqrt{\sum_{k=1}^{d}(v_{i-2,k} - v_{j-2,k})^2} \tag{5.12}$$

where d is the dimension of the history vector.

5.4.3 Algorithm Implementation

The options discussed above determine which paths to merge among different histories based on the prefix tree. However, a lattice is a more common data structure for a compact representation of multiple paths. Hence, in practice, it turns to expend the path for each node in the lattice, until the histories are equivalent according to either Equation 5.9 or 5.10.

It is worth pointing out, during RNNLM lattice rescoring, RNNLMs are applied on the the existed lattice generated by n-gram LMs, instead of prefix tree. The n-gram based clustering and history vector based clustering described in Section 5.4.1 and 5.4.2 are used to determine whether two history paths are equivalent or not. If not, the node will be expended. The extreme case is all merging node are expended and the prefix tree is generated. However, it won't generate new path in the new lattice compared with the input n-gram lattice.

All the lattice rescoring experiments in this thesis use an on-the-fly lattice expansion algorithm [141] suitable for a wide range of language models including back-off n-grams, feedforward NNLMs, recurrent NNLMs and their interpolated form [139]. A central part of this algorithm requires the LM state representation for the underlying model being used. For example, for back-off n-gram LM, this is given by Equation 2.22. For approximated RNNLMs, this is based on Equation 5.9 or 5.10 depending on the history clustering technique being used. The interpolated LM's state representation is derived from a union of those component LMs. The corresponding pseudo-code for the algorithm is given below [5].

1: **for** every node n_i in the network **do**
2: initialise its expanded node list $N'_i = \{\}$;
3: initialise its expanded outbound arc list $A'_i = \{\}$;
4: **end for**
5: add n_0 to its expanded node list, $N'_0 = \{n_0\}$;
6: add all n_0's outbound arcs to its expanded arc list, $A'_0 = A_0$;
7: Start depth first network traversal from the initial node n_0;
8: **for** every node n_i being visited **do**
9: **for** every expanded node $n'_j \in N'_i$ of node n_i **do**
10: **for** every outbound arc a_k from n_i **do**
11: find the destination node n_k of arc a_k;
12: find the LM state $\Phi(h_{n_0}^{n'_j})$ of expanded node n'_j;
13: compute LM probability $P(n_k|\Phi(h_{n_0}^{n'_j}))$;

[5] The pseudo-code is from [144] provided by Xunying Liu

find a new LM state $\Phi(h_{n_0}^{n_k})$ for node n_k;
if $\exists$ node $n'_l \in N'_k$ representing state $\Phi(h_{n_0}^{n_k})$ **then**
return the found node n'_l;
else
add a new node n'_l to N'_k to represent state $\Phi(h_{n_0}^{n_k})$;
end if
create a new arc a'_l from n'_j to n'_l;
assign score $\ln P(n_k|\Phi(h_{n_0}^{n'_j}))$ to a'_l;
add arc a'_l to the expanded outbound arc list A'_i.
end for
end for
end for
Rebuild new network using $\{N'_i\}$ and $\{A'_i\}$.

The above on-the-fly lattice expansion algorithm is implemented as an extension of HLRescore in HTK toolkit and also incorporated into the latest HTK v3.5 [252].

5.5 Experiments

In this section the performance of the proposed RNNLM lattice rescoring methods are evaluated using two HTK-based large vocabulary speech recognition systems. The first was developed for English conversational telephone speech (CTS) used in the 2004 DARPA EARS evaluation [64]. The second system for Mandarin Chinese conversational speech was used in the 2014 DARPA BOLT evaluation [143]. A series of experiments were conducted on these two tasks.

5.5.1 Experiments on English CTS Data

The 2004 CU English CTS LVCSR system was also used for experiments in Chapter 4. A pruned interpolated 3-gram LM was used for lattice generation and followed by lattice rescoring with an unpruned 4-gram LM. The 3 hour **dev04** data, which includes 72 Fisher conversations and contains on average 10.8 words per segment, was used as a test set. The 3 hour **eval04** set of a comparable number of Fisher conversations was also used. For all results presented in this chapter, matched pairs sentence-segment word error (MAPSSWE) based statistical significance test was performed at a significance level $\alpha = 0.05$.

The baseline 4-gram back-off LM was trained using a total of 545 million words from 2 text sources: the LDC Fisher acoustic transcriptions, **Fisher**, of 20 million words (weight 0.75), and the University Washington conversational web data [24], **UWWeb**, of 525 million words (weight 0.25). The **Fisher** data of 20M words contains on average 12.7 words per sentence, and was used to train a feedforward 4-gram NNLM using the OOS architecture proposed in [172], and an RNNLM using the comparable class-based OOS architecture in Figure 4.2 of Chapter 4 with 500 output layer classes. The same 38k word input layer vocabulary and 20k word output layer shortlist were used for both feedforward and recurrent

NNLMs both with 500 hidden layer nodes. A total of 1 billion words of text data were generated from this RNNLM using the sampling technique described in [59] to train a 4-gram back-off LM as an approximation to the original RNNLM. These three LMs were then interpolated with the baseline 4-gram LM and used for Viterbi. Confusion network (CN) decoding [145] is used on lattice for better performance.

LM	N best	PPL	1best	CN	LatDensity
4glm		51.8	16.7	16.1	421
+NN.4g		50.0	16.3	15.8	555
+RNN.sample.4g		50.9	16.2	15.9	462
+RNN	50	46.3	15.4	15.4	188(97)
	100		15.3	15.3	365(175)
	1000		15.3	15.1	3416(1298)
	10000		15.3	15.0	32277(10212)

Table 5.1 Performance of 4-gram LM approximation of RNNLM by sampling and N-best rescoring on the English CTS **dev04** task

Table 5.1 gives the experimental results of the 1-best and CN word error rates (WER) of the baseline back-off 4-gram LM (the 1st line), the feedforward NNLM system (the 2nd line), and the approximated 4-gram LM of RNNLM trained on sampling sentences (the 3rd line) on the **dev04** set. The RNNLM system was evaluated by re-ranking N-best lists of various depth from top 50 up to 10k entries, as shown from 4th to 7th line. The RNNLM rescored N-best lists were also converted to prefix tree structured lattices [211] and used for CN decoding. The HTK formatted lattice density (Arcs/Sec) measure for the baseline systems are also shown in the Table 5.1. For the RNNLM N-best rescoring results, the lattice density measures before and after prefix tree structuring of N-best lists are both given. As expected, the prefix tree structuring of N-best lists significantly reduced the size of the N-best lists (shown in brackets in the same column). As CN decoding favours a more efficient lattice representation that encodes rich alternative hypotheses. To achieve the same improvements from CN decoding, RNNLM rescored N-best list needs to be as deep as 10k. This 10k-best RNNLM rescoring baseline gave the lowest 1-best and CN WER of 15.3% and 15.0% on **dev04** set, with a density of 10.2k arcs/sec measured on the lattices after converting the N-best list to the prefix tree.

The performance of lattice rescoring using RNNLMs is given in Table 5.2. The results of the *n*-gram approximation based RNNLM lattice rescoring methods are given in the first block. When the truncated history is increased to the previous 5 words, the resulting 6-gram approximated RNNLM system produced 1-best and CN error rates of 15.4% and 15.0% on **dev04** set, comparable with the standard RNNLM 10k-best rescoring baseline, and a significant 70% reduction in lattice size from 10k to 3k arcs/sec. Further increasing the truncated history length to 6 words via a 7-gram approximation gave no further improvement while only increasing the size of the resulting lattices. This confirms the hypothesis raised in

Chapter 5.4 over the decaying effect from remote history contexts on RNNLM probabilities.

history clustering	config.	PPL	1best	CN	LatDensity
4glm baseline		51.8	16.7	16.1	421
n-gram	3	46.4	15.8	15.4	428
	4	46.3	15.7	15.2	555
	5	46.3	15.6	15.1	1266
	6	46.3	**15.4**	**15.0**	**3025**
	7	46.3	15.4	15.0	7140
history vector distance	0.00450	46.4	15.8	15.4	465
	0.00300	46.3	15.6	15.2	539
	0.00200	46.3	15.6	15.1	699
	0.00100	46.3	15.6	15.1	1345
	0.00075	46.3	15.5	15.1	1842
	0.00050	46.3	**15.4**	**15.0**	**2818**
	0.00025	46.3	15.4	15.0	4725
	0.00001	46.3	15.4	15.0	6836

Table 5.2 Performance of RNNLM lattice rescoring using *n*-gram based history and history vector distance based clustering on **dev04** set

The results of the hidden history vector distance based RNNLM lattice rescoring are shown in the bottom section of Table 5.2. By adjusting the hidden vector distance beam γ in Equation 5.10, a range of approximated RNNLM comparable in error rates with the truncated history based approach but more compact lattices were produced. For example, setting $\gamma = 0.002$ produced equivalent 1-best and CN error rates of 15.6% and 15.1% as the 5-gram history approximated system for **dev04 set**, and a 45% reduction in lattice size from 1266 down to 699 arcs/sec. The best performance was obtained by setting $\gamma = 0.00050$, which gave 1-best and CN error rates of 15.4% and 15.0%, with a 72.4% and 7% reduction in lattice size over the 10k-best rescoring baseline, and the best 6-gram history clustering rescoring system respectively. In practice, a system setting of history vector distance to 0.00050 can be used to rescore more heavily pruned lattices at 0.9 time real time (RT) while producing comparable 1best and CN error rates of 15.4% and 15.1%. In contrast, the 1k-best and 10k-best rescoring systems used 1.8 and 17 times RT respectively. The similar trend was observed on **eval04** set, which is given in Table 5.3.

5.5.2 Experiments on Mandarin CTS Data

The 2014 CU CTS Mandarin Chinese LVCSR system [143] was used to further evaluate the two proposed RNNLM lattice rescoring methods. The system was trained on 300 hours of Mandarin Chinese conversational telephone speech data released by the LDC for the DARPA BOLT program. A 63k recognition word list was used in decoding. The system

LM type	usage	config.	PPL	1best	CN	LatDensity
4glm			52.1	19.1	18.7	430
+NN.4g			50.9	18.7	18.2	574
+RNN.4g	sample		51.1	18.9	18.4	472
+RNN	Nbest	50	46.6	17.9	17.9	200(98)
		100		17.9	17.7	389(177)
		1000		17.8	17.6	3607(1313)
		10000		17.8	17.5	33607(10275)

Table 5.3 Performance of N-best rescoring and sampling with RNNLM on the English CTS task on **eval04**

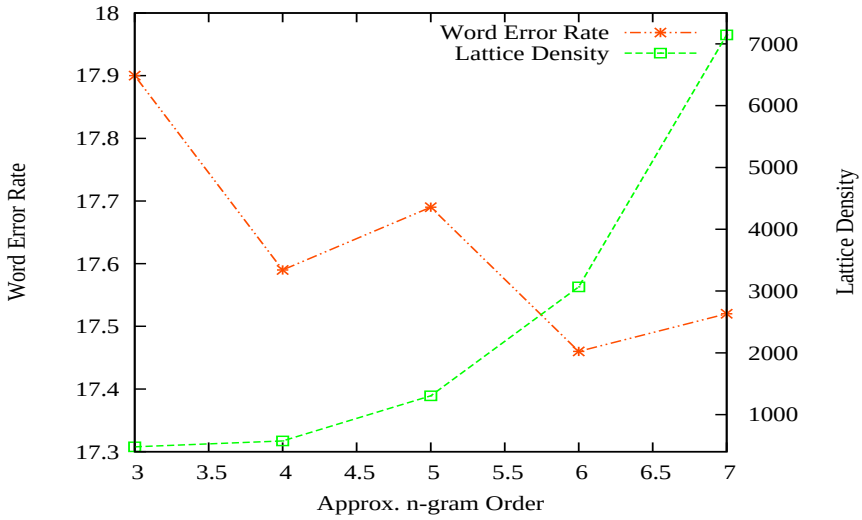


Fig. 5.7 *WER and lattice density for n-gram approximation based RNNLM lattice rescoring on the English CTS task on* **eval04**

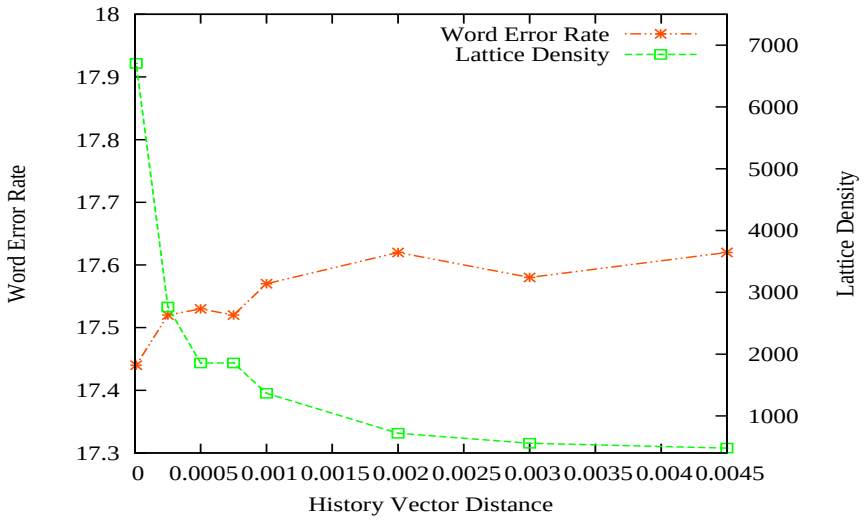


Fig. 5.8 *WER and lattice density for history vector distance based RNNLM lattice rescoring on the English CTS task on* **eval04**

uses the same multi-pass recognition framework but with a more advanced acoustic model compared the the above experiment.

The initial lattice generation stage used CMLLR [68] based speaker adaptively trained cross-word triphone Tandem [95] HMM acoustic models with MPE [178] based parameter estimation and unsupervised MLLR [133] speaker adaptation. HLDA [127, 137], projected and speaker level normalised PLP [241] features augmented with pitch features were used. 26 dimensional DNN bottle neck features [253] extracted from a deep neural network [51, 198] consisting of 4 hidden layers of 1k nodes each and modelling 6k context dependent states at the output layer, were also used. An interpolated 4-gram baseline LM was used. A 4.5 hour test set of Mandarin Chinese conversational telephone speech data used in the BOLT program, **dev14**, consisting of 57 speakers from 19 conversations, was used for performance evaluation. An additional 1.6 hour test set, **eval97**, consisting of 49 speakers from 20 conversations, was also used. Manual audio segmentation was also used to allow translation outputs to be accurately scored.

The baseline 4-gram back-off LM was trained using a total of 1 billion words from the following two types of text sources: 2.6M words of acoustic transcripts including the LDC Call Home Mandarin (CHM), Call Friend Mandarin (CFM) and HKUST collected conversational Mandarin telephone speech data (weight 0.78); 1 billion words of additional web data collected under the DARPA EARS and GALE programs (weight 0.22). The acoustic transcripts contain on average 7.5 words per sentence. This baseline 4-gram LM has a total of 48M 2-grams, 133M 3-grams and 143M 4-grams. It gave a perplexity score of 151.4, 1-best and CN character error rates (CER) of 35.7% and 35.3% respectively on **dev14**, 140.0, 31.3 and 31.1 on **eval97**. These results are shown in the 1st line in Table 5.4.

In order to further improve the RNNLM's coverage and generalisation, the 2.6M words of acoustic transcripts data were augmented with 15M words of its paraphrase variants.

These were automatically produced using the statistical paraphrase induction and generation method described in [140]. The above combined data set was then used to train a paraphrastic RNNLM [136] "rnn" on a GPU in bunch mode [40]. The full output layer with an OOS node based RNNLM architecture in Figure 3.2 was used. A total of 512 hidden layer nodes were used. A 27k word input layer vocabulary and 20k word output layer shortlist were also used. In common with the previous experiments of Chapter 5.5.1, a total of 1 billion words of text data were also generated from the RNNLM "rnn" using the same sampling technique described in [59] to train a 4-gram back-off LM as an approximation. Both the RNNLM and the sampled data trained 4-gram LM were then interpolated with the baseline 4-gram LM for performance evaluation. The perplexity, 1-best and CN decoding CER performance of the baseline RNNLM and various approximation schemes are shown in Table 5.4. Consistent with the trend previously found in Table 5.1, the sampling approach based RNNLM approximation (line 2 in Table 5.4) only retained a part of the improvement of the original RNNLM (lines 3 to 6 in Table 5.4) over the baseline 4-gram LM in terms of both perplexity and error rate. Using the prefix tree structured N-best lists again significantly reduced the density of the resulting lattices. The best CN decoding performance was obtained using a 10k-best RNNLM rescoring baseline system. On the **dev14** data, it gave a 1-best and CN CER of 34.6% and 34.3%. It had a density of 11k arcs/sec measured on the lattices converted from the prefix tree structured 10k-best lists.

LM type	N best	PPL	1-best	CN	LatDensity
4glm		151.4	35.7	35.3	273
+RNN.sample.4g		140.6	35.2	34.8	310
+RNN	50	127.1	34.7	34.6	185(102)
	100		34.6	34.5	360(187)
	1000		34.5	34.4	3329(1417)
	10000		34.6	34.3	30597(11007)

Table 5.4 Performance of N-best rescoring and sampling with RNNLM on Mandarin CTS **dev14** testset

The results of RNNLM lattice rescoring on the **dev14** set are given in Table 5.5. The performance of the *n*-gram history clustering based RNNLM lattice rescoring of Chapter 5.4 is shown from in the first block. A 6-gram approximate RNNLM system produced 1-best and CN error rates of 34.7% and 34.2% respectively on **dev14**. Both results are comparable to the 10k-best RNNLM rescoring in Table 5.4. It also gave a significant 74% reduction in lattice density from 11k to 2852 arcs/sec. Further increasing the truncated history to 6 words or more gave no improvement but only increasing the resulting lattice size. The performance of the hidden history vector distance based RNNLM lattice rescoring is shown in the bottom block of Table 5.5. The hidden vector distance beam $\gamma = 0.00195$ gave the best CER performance among all systems. This approximate system gave a 1-best and CN error rates of 34.6% and 34.2% respectively. It also gave a 68.2% relative reduction in lattice

density over the prefix tree structured 10k-best rescoring system in Table 5.4 from 11k down to 3501 arcs/sec.

history clustering	config.	PPL	1-best	CN	LatDensity
4glm baseline		151.4	35.7	35.3	273
n-gram	3	127.5	34.8	34.5	305
	4	127.1	34.8	34.4	554
	5	127.1	34.7	34.3	1285
	6	127.1	34.7	34.2	2852
	7	127.1	34.6	34.3	6012
	8	127.1	34.7	34.2	12695
history vector distance	0.00900	127.3	34.8	34.5	500
	0.00800	127.1	34.8	34.4	564
	0.00700	127.1	34.9	34.3	658
	0.00600	127.1	34.8	34.4	802
	0.00500	127.1	34.8	34.3	1034
	0.00400	127.1	34.7	34.3	1430
	0.00300	127.1	34.7	34.3	2112
	0.00195	127.1	34.6	34.2	3501
	0.00185	127.1	34.6	34.2	3705
	0.00175	127.1	34.6	34.2	3905
	0.00150	127.1	34.6	34.2	4427
	0.00100	127.1	34.6	34.2	5652
	0.00010	127.1	34.6	34.2	8098
	0.00001	127.1	34.6	34.2	8215

Table 5.5 Performance of RNNLM lattice rescoring using *n*-gram based history and history vector distance based clustering on Mandarin CTS **dev14** testset.

A similar trend was also found on the **eval97** data in Table 5.6.

5.5.3 Experiments on Babel Corpus

The next experiment is conducted on the Babel corpora [72, 50], which consist of transcribed telephone conversations in a range of languages for speech recognition and keyword search (KWS). The aim of the Babel Program is "developing agile and robust speech recognition technology that can be rapidly applied to any human language in order to provide effective search capability for analysts to efficiently process massive amounts of real-world recorded speech" [6]. Keyword search, also known as spoken term detection, is a speech processing

[6] quoted from https://www.iarpa.gov/Programs/ia/Babel/babel.html. The corpora ids used in this section in the Babel language releases are Pashto (104) IARPA-babel104b-v0.4bY, Igbo (306) IARPA-babel306b-v2.0c, Mongolian (401) IARPA-babel401b-v2.0b, Javanese (402) IARPA-babel402b-v1.0b and Georgian (404) IARPA-babel404b-v1.0a.

LM type	usage	config.	PPL	1-best	CN	LatDensity
4glm			140.0	31.3	31.1	273
+RNN	sample		135.0	31.2	30.9	326
+RNN	nbest	50	125.0	30.5	30.5	238(128)
		100		30.5	30.5	458(232)
		1000		30.5	30.4	3861(1610)
		10000		30.5	30.4	31423 (10848)

Table 5.6 Performance of N-best rescoring and sampling with RNNLM on the Mandarin CTS **eval97** testset.

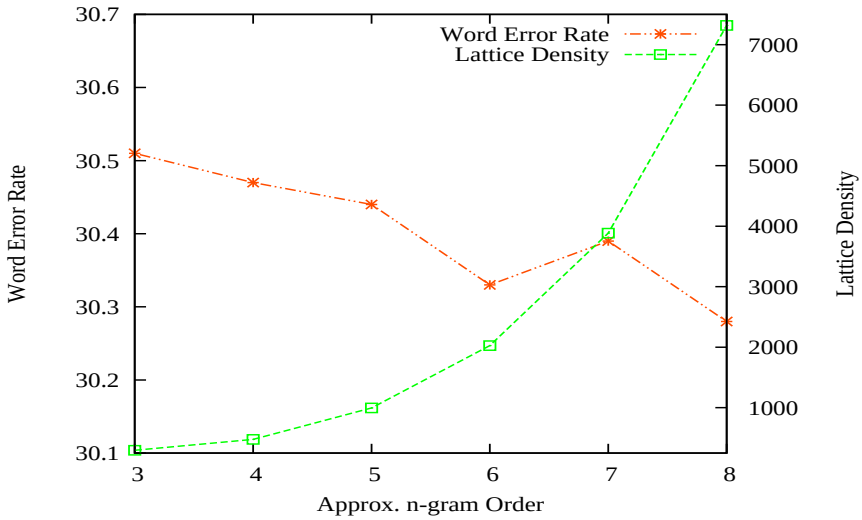


Fig. 5.9 *WER and lattice density for n-gram approximation based RNNLM lattice rescoring on the Mandarin CTS* **eval97**.

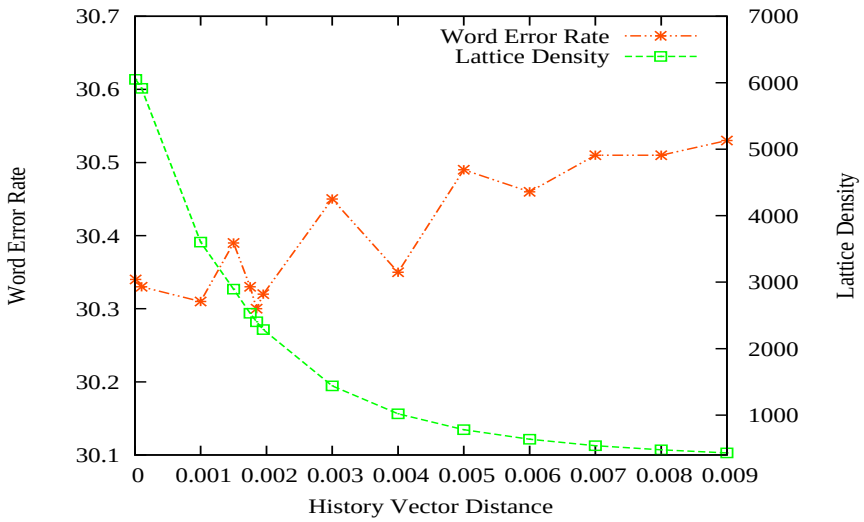


Fig. 5.10 *WER and lattice density for history vector distance based RNNLM lattice rescoring on the Mandarin CTS* **eval97**.

task to find all occurrences of a word or word sequence, in a large audio corpus [122, 117]. Figure 5.11 gives an example framework of keyword search system. Lattices are generated from the ASR system and the query (i.e. keyword) is searched among all possible paths in lattices. The posterior of the keyword in lattices can be computed based on their acoustic model and language model probabilities [237].

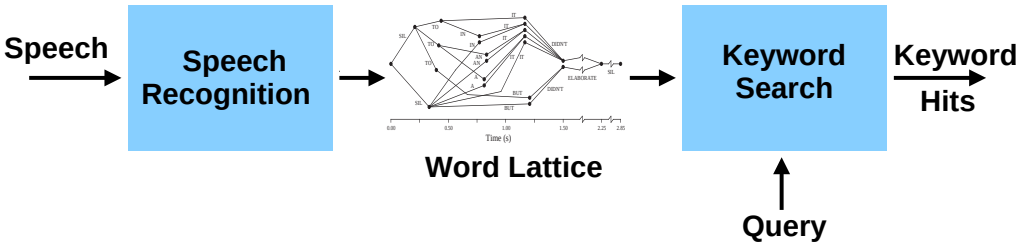


Fig. 5.11 *A framework of keyword search system.*

Given the keyword list containing the set of keywords of interest, two types of errors are defined in the keyword search system: miss error and false alarm error. The miss error refers to the case that the keyword indeed appears in the speech but the KWS system doesn't spot it. The false alarm error means that there is no keyword in the speech while the KWS system outputs it as a keyword. The rates of miss and false alarm errors can be computed as

below,

$$P_{miss}(w_{kw},\theta) = 1 - \frac{N_{corr}(w_{kw},\theta)}{N_{ref}(w_{kw})}$$
$$P_{fa}(w_{kw},\theta) = \frac{N_{incorr}(w_{kw},\theta)}{N_{trial}(w_{kw})} \tag{5.13}$$

where $N_{ref}(w_{kw})$ is the number of reference occurrence of keyword w_{kw}, $N_{corr}(w_{kw},\theta)$ is the number of correctly hypothesised occurrence of w_{kw} at the detection threshold θ, $N_{incorr}(w_{kw},\theta)$ is the number of incorrectly hypothesised occurrence of w_{kw} at threshold θ, and $N_{trial}(w_{kw})$ is the number of trials for w_{kw}. $N_{trial}(w_{kw})$ is defined by,

$$N_{trial}(w_{kw}) = T_{speech} - N_{ref}(w_{kw}) \tag{5.14}$$

where T_{speech} is the total speech duration in seconds and a rate of one trial per second is assumed [117].

In the spoken term detection (STD) 2006 evaluation, a metric, "term-weighted value" (TWV), is defined by NIST as,

$$TWV(\theta) = 1 - \frac{1}{|Q|}\sum_{w_{kw}\in Q}(P_{miss}(w_{kw},\theta) + \beta P_{fa}(w_{kw},\theta)) \tag{5.15}$$

where $|Q|$ is the number of keywords in the keyword list Q, and $\beta = 999.9$ to reflect the relative cost of miss and false alarm errors. The value of TWV(θ) lies in the range of $(-\infty, 1.0]$. The reference of keyword gives a TMW of 1.0 and an empty output is corresponding to 0.0 for TMW. The maximum term-weighted value (MTWV) [7] is used in the the experiment, which is the best term-weighted value with an optimised detection threshold θ.

The full language pack (FLP) in the Babel Program has about 100-200 hours of transcribed audio training data ($\sim$60-80 hours speech) for the training of acoustic model. Tandem and Hybrid systems are built with speaker adaptation using CMLLR transferred feature [68]. To obtain better performance, a total of four acoustic models, two Tandem and two Hybrid systems, are build using HTK toolkit [252], based on the multi-lingual feature [229] provided by IBM and Aachen University. Joint decoding [233] discussed in Chapter 2.2.4 was used to combine these 4 acoustic models. A brief description of joint decoding can be found in Chapter 8.2. The corpus to build language model consists of two sources: one is from the acoustic transcription, which is referred as FLP data; the other one is from the web using a search engine, e.g. Wikipedia, Ted talk, Tweets. The amount of the FLP data is significantly smaller than that of the WEB data. A total of 5 languages were examined in this experiment. The information of these 5 languages is shown in Table 5.7. The vocabulary size varies from 28K in Igbo to 376K in Pahto and the amount of the WEB data also varies from 2M in Igbo to 141M in Mongolian. The size of FLP data is stable among these 5 languages, which lies in the range of 400K to 500K. For each language, the vocabulary size is larger than 200K and the amount of web data is about 100M, except Igbo. The 3-gram

[7] "NIST Tools": http://www.itl.nist.gov/ iad/mig/tools/

LMs were used in this experiment as 4-gram LMs didn't give performance improvement. The 3-gram LMs and RNNLMs were built for the 5 languages and their perplexity results are also given in the table. For the training of RNNLMs, the WEB data is first used. The model trained on the WEB data is then fine-tuned on the FLP data. All RNNLMs are trained with slipced bunch mode on GPU as introduced in Chapter 4. The 3-gram approximation described in Chapter 5.4.1 is used for RNNLM lattice rescoring based on lattices generated by the 3-gram LMs. All RNNLMs shown in Table 5.7 were trained with cross entropy with a single hidden layer with 100 nodes. It can be seen that RNNLMs gave significant improvements over 3-gram LMs in terms of perplexity on each language.

Table 5.7 Statistics and perplexity results for 5 languages in the Babel corpora.

Language	Vocab Size	#Train Word		PPL	
		WEB	FLP	3-gram LM	+RNNLM
Pashto	376,271	104M	535K	171.6	135.8
Igbo	28,097	2M	549K	109.9	94.3
Mongolian	246,831	141M	512K	133.6	105.4
Javanese	268,099	73M	409K	218.8	172.1
Georgian	278,623	137M	406K	472.4	377.2

Pashto is used to investigate the WER and MTWV performance first and Table 5.8 gives the WER results of the 3-gram LM and RNNLM on Pashto. RNNLM gave a reduction of 1.0% in WER. However, in the RNNLM for Pashto, an output shortlist consisting of 282k words is used. The CE training is very slow due to the large output layer, which takes up to one week. Furthermore, a full output layer RNNLM was trained from CE based criterion and the evaluation is also time-consuming as the explicit normalisation in the large output layer is required.

Table 5.8 WER results of RNNLM on Pashto.

LM	WER
3-gram	43.8
+RNN	42.8

One solution to improve the evaluation efficiency is adopting RNNLMs with class based output layer. However, as discussed in Chapter 4, it is difficult to parallel the training of class based RNNLMs. In this section, various training criteria described in Chapter 4 are investigated. As mentioned above, there are two stages for the RNNLM training, which are on the WEB and FLP data successively. Noise contrastive estimation (NCE) [44] is suitable for the training of the WEB data. However, there is an issue for the fine-tune on the FLP data. Given the large output layer size and small amount of FLP data, there are many words in the output layer that do not appear in the FLP data, and the unigram probability of these words turns to be 0. As a result, the weight associated with these words in the output layer

won't be trained as target word or noise word during the NCE training on the FLP data. Poor performance was obtained by using this unigram for NCE training[8].

Three methods were proposed to address the above issues, and their WER and MTWV results are presented in Table 5.9. The first approach applies CE on the WEB data, and variance regularisation (VR) [43] on the FLP data for fine-tuning. The model trained with variance regularisation was used for lattice rescoring without normalisation in the output layer, which speeded up the evaluation significantly. The WER and MTWV results are shown in the 3rd line of Table 5.9. It gave a small degradation in terms of performance compared to RNNLMs trained with CE in the 2nd line, while much better than the 3-gram LM shown in the 1st line. However, it took long time to train the CE based RNNLM on the WEB data. The second method is to apply NCE training on the WEB data and then adopt VR on the FLP data. In addition to solve the evaluation efficiency issue by using RNNLM trained with VR, it also has a much faster train speed on the large amount of WEB data by NCE training, which only takes about 6 hours for training. The results can be found in the 4th line of Table 5.9. It gave a comparable WER and MTWV performance as the RNNLM trained with CE, while much faster for RNNLM train and evaluation. The third method aims to use NCE for the whole training, in order to avoid the unigram issue in the NCE training mentioned above: 9 copies of the FLP data are appended to the end of the WEB data to construct a "extended" corpus, and the NCE training is then applied to this "extended" corpus. The results are shown in the last line in the table, which gave fast train and evaluation, but with a degraded performance.

Table 5.9 WER and KWS results of RNNLM trained with various criteria on Pashto.

LM	Train Crit		WER	MTWV		
	WEB	FLP		IV	OOV	Total
3-gram	-		43.8	0.4828	0.4083	0.4750
+RNN	CE	CE	42.8	0.4975	0.4048	0.4871
	CE	VR	43.0	0.4958	0.4010	0.4853
	NCE	VR	43.0	0.4975	0.3953	0.4862
	NCE	-	43.2	0.4936	0.4038	0.4835

Considering the balance of performance and efficiency, the second method (NCE+VR) provides the best solution among the three approaches. This method was then applied to the remaining 4 languages. The WER and KWS performance can be found in Table 5.10. The RNNLMs were trained efficiently (all less than 8 hours) and consistent improvements were obtained on the 4 languages in terms of WER and MTWV.

[8]Unigram from the WEB data or interpolated with the unigram from WEB data were also tried, which gave degraded performance

Table 5.10 WER and KWS results of RNNLM trained on different languages.

Language	LM	WER	MTWV		
			IV	OOV	Total
Igbo	3-gram	54.6	0.4079	0.3635	0.4030
	+RNN	53.7	0.4101	0.3718	0.4061
Mongolian	3-gram	47.0	0.5606	0.5171	0.5559
	+RNN	46.0	0.5708	0.5343	0.5666
Javanese	3-gram	50.1	0.5182	0.4801	0.5138
	+RNN	49.3	0.5229	0.4768	0.5173
Georgian	3-gram	37.8	0.7325	0.7300	0.7322
	+RNN	37.1	0.7386	0.7309	0.7375

5.6 Summary

In this chapter, lattice rescore using RNNLM is studied. Two efficient lattice rescoring methods for RNNLMs were proposed. Unlike previous work which generated an approximated *n*-gram LM or only rescored the top N-best lists, RNNLM probability is used to rescore the whole lattice. The proposed techniques produced 1-best and confusion network decoding performance comparable with a 10k-best rescoring RNNLM baseline systems on two large vocabulary conversational telephone speech recognition tasks for US English and Mandarin Chinese. These methods also produced highly compact lattice representation after RNNLM rescoring. Consistent compression in lattice size was obtained over the prefix tree structured N-best rescoring RNNLM baseline systems. These results demonstrate the advantages of the proposed techniques over the traditional methods to incorporate RNNLM, such as N-best rescoring and *n*-gram LM approximation through sentence sampling. Consistent improvements are also obtained for the keyword search system by using lattices generated by RNNLM lattice rescoring.

Chapter 6

Adaptation of RNNLMs

Acoustic model adaptation is a crucial technique in speech recognition to mitigate the acoustic mismatch that may exist between training and test data, such as unexpected speaker, background noise and channel. Significant improvement in performance has been reported from acoustic model adaptation. Mismatch can also occur in the nature of language to be modelled. Normally the training corpus for a language model is collected from a variety of sources with different content, written or speaking form, genre or topic. The style of corpus can heavily influence words to be used. When applied to unseen test data, mismatch arises in various aspects including speaking style and topic, which degrades performance as a result. Hence, language model adaptation is an important research area and very useful for practical application. Approaches for adapting *n*-gram LMs was discussed in Chapter 2.3.6. In this chapter, the adaptation of neural network based language models will be detailed. There are two approaches to adapt RNNLMs described in the literature: fine-tuning; and incorporation of informative feature. In this chapter, a detailed study of RNNLM adaptation is carried out on a multi-genre broadcast news task using a strong ASR system, where the adaptation can be operated at different levels, such as genre and show level. A range of topic models are used to extract topic representation for each show.

This chapter is organised as follows. RNNLM adaptation methods are reviewed in Chapter 6.1. Genre dependent RNNLM adaptation is described in Chapter 6.2, followed by topic representation on show level for adaptation in Chapter 6.3. The experimental results are presented in Chapter 6.4. Finally, Chapter 6.5 concludes this chapter.

6.1 Review of RNNLM Adaptation

A standard framework for unsupervised RNNLM adaptation for speech recognition is illustrated in Figure 6.1 [151, 45]. The blue part is the conventional speech recognition without language model adaptation and the red part denotes the pipeline for RNNLM adaptation. 1-best hypothesis and lattice are generated during searching given the input utterance. The 1-best hypothesis is used to adapt RNNLM and the adapted RNNLM obtained. The adapted RNNLM can be used to rescore lattice generated from the ASR system and the adapted 1-best hypothesis obtained. The gray block in Figure 6.1 gives an example of RNNLM adap-

tation using the output of speech recogniser and there are other options to adapt RNNLMs. For example, the adapted RNNLM can be obtained by fine-tuning the model on in-domain data and applied directly on lattice.

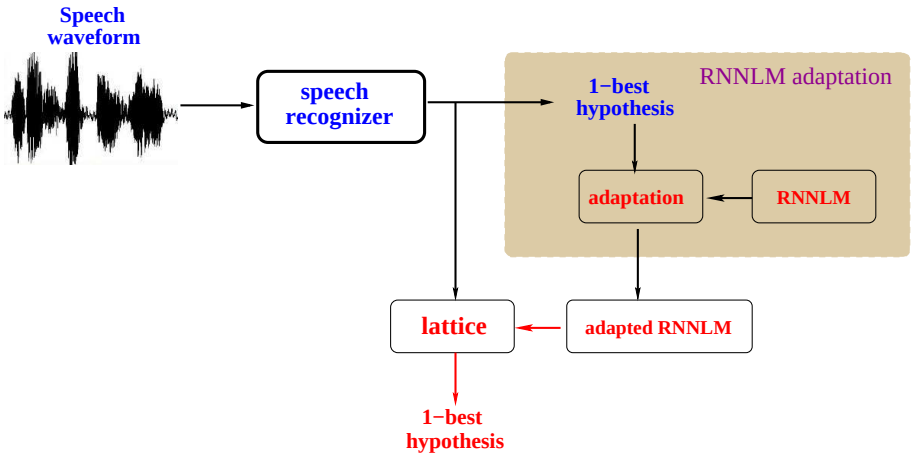


Fig. 6.1 *A standard framework for unsupervised RNNLM adaptation in speech recognition.*

Adaptation methods of RNNLMs are divided into two broad categories in the literature, which are fine-tuning and incorporation of informative features.

6.1.1 Fine-tuning

RNNLMs are initially trained on all available training data. If this initial RNNLM is to be applied to a specific target domain, then fine-tuning with the in-domain data can be applied. These in-domain data may come from a subset of training data, where the prior information about the domain (or genre) of data is available. The adapted model can then be applied to the corresponding test data according to their domain (i.e. genre) labels [206, 172]. In many applications, the reference of adaptation data is not easily available. The hypothesis from ASR is used instead for fine-tuning to improve the model [123] in an unsupervised mode. Depending on the amount and accuracy of the adaptation data, the whole or partial model parameters are updated. Figure 6.2 illustrates the fine-tuning of RNNLMs.

Fine-tuning provides a straightforward way to tune the well-trained universal model into specific domain. If the prior information (e.g. domain label) and sufficient amount of adaptation data are available, the model could be improved by adaptation and obtain better performance. However, in many situations, sufficient in-domain data and their domain information are difficult to obtain. Furthermore, it requires storing one model for each domain, which requires large disk space when a range of domains are used.

6.1.2 Incorporating Auxiliary Feature

An alternative approach to adapt RNNLMs is to incorporate auxiliary features, which carry information about the speech and topics. This informative feature could be incorporated

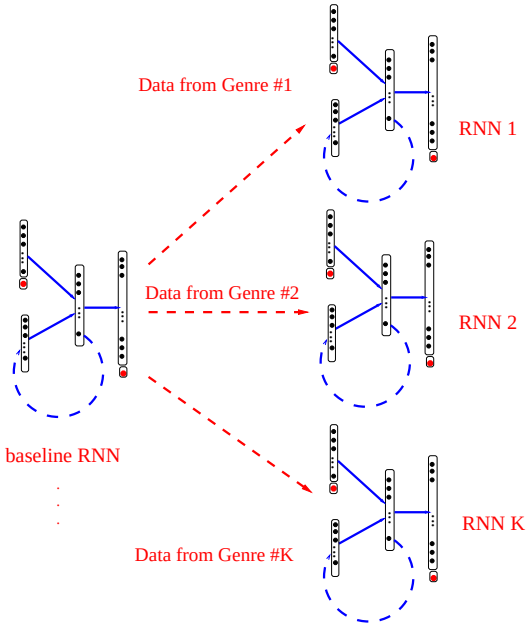


Fig. 6.2 *RNNLM adaptation with genre information based on fine-tuning.*

into the training of RNNLMs for adaptation in a more compact and efficient way compared to fine-tuning. Figure 6.3 illustrates the adaptation scheme. The feature vector $\mathbf{f}$ is appended to the input layer. It is fed into the hidden layer and output layer[1] as introduced in [152].

A range of auxiliary features have been investigated for adaptating RNNLMs in earlier research. For example, morphological and lexical features have been modelled in factored RNNLMs [244] on the 930k word WSJ portion of Penn Treebank data; topic information derived from latent Dirichlet allocation (LDA) [18] models was used in [151] on a corpus of 37 million words; personalized user information such as demographic features was exploited in [235] for RNNLMs on a social media corpus of approximately 25 million words; sentence length information and lexical features were used in [204] on lecture transcripts of 9 million words; and domain information was used in mutli-domain RNNLMs [227] on a 10 million word medical report corpus.

There are two important issues that directly impact the auxiliary feature based RNNLM adaptation approach: the form of input feature representation to use; and the scalability when larger amounts of training data are used. In this chapter, both of these issues are explored. Genre and topic based RNNLM adaptation techniques are investigated on a multi-genre BBC broadcast transcription task. The BBC provided us with genre information for each broadcast show and this information is used for adaptation. A range of techniques including LDA, probabilistic latent semantic analysis (PLSA) [99] and hierarchical Dirichlet processes (HDP) [226] are used to extract a show-level topic representation as continuous

[1] According to our experimental results, the connection between input (block $\mathbf{f}$) and output layer is crucial when the hidden layer size is small (e.g. < 50). When the size of hidden layer becomes large (e.g. > 100), there is no difference between using and not using the connection with the output layer. In this work, the output layer connection is used.

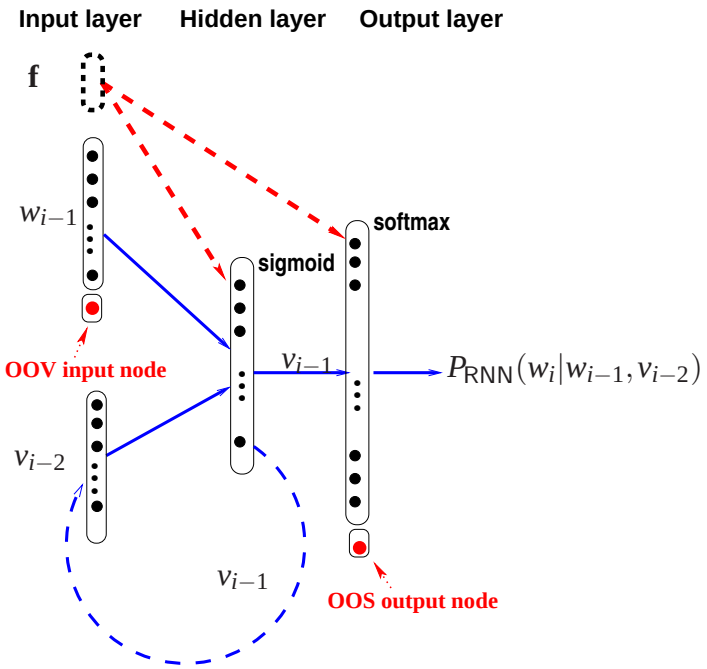


Fig. 6.3 *An example RNNLM with an additional input feature vector* $\boldsymbol{f}$.

valued vectors. These additional topic vectors are used for RNNLM training and then to facilitate adaptation at test time.

6.2 Adaptation using Genre Information

In some applications, the text data for training contains a mix of different genres, and genre information is available, such as the broadcast news data where the genre information (e.g. news, sports, documentary) can be obtained easily. RNNLMs can be refined by making use of this genre information. The first and most straightforward way is to fine-tune a well-trained genre-independent RNNLM using genre-specific data as discussed in Chapter 6.1.1. This yields a set of genre-dependent RNNLMs, one for each genre. At test time, for each utterance, the genre-specific RNNLM is applied according to its genre label. The potential drawbacks of this method are that a RNNLM for each genre needs to be stored and sufficient data for each genre is required for robust training. An alternative approach to construct genre-dependent RNNLMs is to incorporate the genre label into the training of RNNLM as discussed in Chapter 6.1.2. The genre label can be represented as a 1-of-k encoding feature vector in the input layer as shown in Figure 6.3. Compared to the fine-tuning scheme, only one RNNLM model is trained and stored. Different genre models are obtained by using the 1-of-k encoding vectors. Both of these approaches yield genre-level adapted RNNLMs. Their performances are investigated and compared in Chapter 6.4.

However, in many situations, the genre label is unknown and possibly difficult to robustly estimate. Furthermore, the genre label is normally a coarse representation of the type of topic. It may be not able to classify data accurately. Hence, a more refined representation

is preferred to automatically derive a topic representation for each show (i.e. document). This show-level topic representation **f**, will be concatenated with the standard input layer for RNNLM training and testing as shown in Figure 6.3.

6.3 Topic Space based Adaptation

Contextual factors, such as speaking style, genre and topic heavily influence the use of word in spoken language. A complex combination of these factors define a specific target situation of interest. The variabilities introduced by these hidden factors are only implicitly learned in conventional RNNLMs. Since it is problematic to draw upon related and similar events occurring in the training data, direct adaptation of RNNLM parameters given limited data at test time to a target situation is difficult to obtain good generalisation performance. One solution to this problem is to explicitly model these influencing factors during RNNLM training, for example, by adding auxiliary features into the input layer. This allows RNNLMs to better exploit commonalities and specialties among diverse data. It also facilitates adaptation at test time to any target situation defined by these factors.

Various topic models have been proposed for topic representation of documents, including probabilistic latent semantic analysis (PLSA), latent Dirichlet allocation (LDA) and hierarchical Dirichlet processes (HDP). Both PLSA and LDA use a fixed number of latent topics. In contrast, HDP is able to estimate the posterior of the number of topics during training and choose the number of topics automatically.

Let $\mathcal{D} = \{d_1, ..., d_N\}$ denote the training corpus, where d_i is a document. $\mathcal{V} = \{w_1, ..., w_V\}$ is the set of all distinct words in the vocabulary, $\mathcal{T} = \{z_1, ..., z_K\}$ is the set of latent topics, and $n(d_i, w_j)$ is the word count of w_j appearing in document d_i. For each document d_i, a vector of posterior probabilities among topics

$$\mathbf{f} = \begin{bmatrix} P(z_1|d_i) \\ \cdots \\ P(z_k|d_i) \\ \cdots \\ P(z_K|d_i) \end{bmatrix}$$

is derived from the specified topic model $\hat{\mathcal{M}}_T$, where each topic has a multinomial distribution over the given vocabulary.

When incorporating the feature vector **f** into RNNLM training as shown in Figure 6.3, a Bayesian interpretation of the RNNLM probability for word w_i given history h_i in a document d' is given by,

$$P_{RNN}(w_i|h_i, \mathcal{D}, d') = \iint P_{RNN}(w_i|h_i, \mathbf{f}) P(\mathbf{f}|\mathcal{M}_T, d') P(\mathcal{M}_T|\mathcal{D}) d\mathbf{f} d\mathcal{M}_T \tag{6.1}$$

where $P(\mathbf{f}|\mathcal{M}_T, d')$ is the topic posterior of d' given a model $\mathcal{M}_T$ trained on corpus $\mathcal{D}$. The exact calculation of the above integral is intractable in general. Hence, approximations are

required to make it feasible. For topic model $\mathcal{M}_T$, a MAP estimate is used instead,

$$\hat{\mathcal{M}}_T = \arg\max_{M_T} P(\mathcal{M}_T|\mathcal{D}) = \arg\max_{M_T} P(\mathcal{D}|\mathcal{M}_T) \tag{6.2}$$

when a uniform prior $P(\mathcal{M}_T)$ is used. A further approximation is made,

$$P(\mathbf{f}|\hat{\mathcal{M}}_T, d') \approx \delta(\mathbf{f} - \hat{\mathbf{f}}_{\hat{\mathcal{M}}_T,d'}), \tag{6.3}$$

the topic posterior $\hat{\mathbf{f}}_{\hat{\mathcal{M}}_T,d'}$ can be obtained by maximising $P(d'|\hat{\mathcal{M}}_T)$.

Hence, the process in Equation 6.1 is be simplified as,

1) train a topic model $\hat{\mathcal{M}}_T$ to maximise the data likelihood in Equation 6.2;
2) computing the topic posterior vector $\hat{\mathbf{f}}_{\hat{\mathcal{M}}_T,d'}$ for document d' given topic model $\hat{\mathcal{M}}_T$.
3) $\hat{\mathbf{f}}_{\hat{\mathcal{M}}_T,d'}$ is used in RNNLM training and applied in the adaptation stage as shown in Figure 6.1.

6.3.1 Probabilistic Latent Semantic Analysis

PLSA [99, 165, 76] is a generative model defined over a given set of documents. Each document is assumed to be generated from a mixture of latent topics $\{z_k\}$ $(k = 1,...,K)$. Each topic z_k is defined by a word distribution $P(w_i|z_k)$. The EM algorithm is applied to maximise the following likelihood criterion,

$$\ln P(\mathcal{D}|\mathcal{M}_T) = \sum_{i=1}^{N}\sum_{j=1}^{M} n(d_i, w_j) \ln \sum_{k=1}^{K} P(z_k|d_i)P(w_j|z_k) \tag{6.4}$$

where $n(d_i, w_j)$ is the count of word w_j occurring in document d_i, $P(z_k|d_i)$ is the probability of the document d_i assigned to topic z_k and $P(w_j|z_k)$ is the word distribution associated with topic z_k. Given a test document d', the topic probability $P(z_k|d')$ is obtained by fixing $P(w_j|z_k)$ and maximising

$$\ln P(d'|\hat{\mathcal{M}}_T) = \sum_{j=1}^{M} n(d', w_j) \ln \sum_{k=1}^{K} P(z_k|d')P(w_j|z_k), \tag{6.5}$$

where $P(z_k|d')$ is found as,

$$\frac{P(d'|z_k)}{\sum_{m=1}^{K} P(d'|z_m)} = \frac{\prod_{j=1}^{M} P(w_j|z_k)^{n(d',w_j)}}{\sum_{m=1}^{K}\prod_{j=1}^{M} P(w_j|z_m)^{n(d',w_j)}}. \tag{6.6}$$

6.3.2 Latent Dirichlet Allocation

LDA [18, 224] adds a prior distribution over the model parameters, $p(\theta;\alpha)$, to relax the constraint of using a fixed set of document level topic posteriors $\{P(z_k|d_i)\}$ in PLSA. Given

a hyper-parameter α, a multinomial parameter distribution $p(\theta;\alpha)$ is defined. LDA is a generative model. For each word w_j in each document d_i, topic z_k is sampled from the topic distribution θ to generate word w_j. The following likelihood is maximised during training,

$$\ln P(\mathcal{D}|\mathcal{M}_T) = \sum_{i=1}^{N} \ln \int \prod_{j=1}^{M} \left(\sum_{k=1}^{K} P(w_j|z_k) P(z_k|\theta) \right)^{n(d_i,w_j)} p(\theta;\alpha) d\theta \tag{6.7}$$

Again where N is the number of document and M is the number of distinct words in vocabulary. The number of topic in LDA is set to K and fixed during both training and inference. The exact posterior inference using LDA is intractable, and a variational approximation or sample based approach can be used instead. A Gibbs sampling based implementation in [176] is used in this work. The posterior probability of each topic z_k given document d' is computed as,

$$P(z_k|d') = \frac{n(d',z_k)+\alpha}{\sum_{m=1}^{K}(n(d',z_m)+\alpha)} \tag{6.8}$$

where $n(d',z_k)$ is the number of words assigned to topic z_k in document d'.

6.3.3 Hierarchical Dirichlet Process

HDP [226] is a nonparametric Bayesian model for clustering problems with multiple groups of data. Its modelling hierarchy consists of two levels. The first level samples the number of topics and topic-specific parameters. The bottom level samples the topic assignment for each word in each document based on the samples drawn from the top level. In PLSA and LDA, the number of topics is chosen empirically, while HDP can estimate the posterior probability over the number of topics. Equation 6.1 can be rewritten by sampling the topic model $\mathcal{M}_T^k$ with k topics from $\mathcal{M}_T^k \sim P(\mathcal{M}_T|\mathcal{D})$ as,

$$P_{RNN}(w_i|h_i,\mathcal{D},d') = \frac{1}{N_{M_T}} \sum_{n=1}^{N_{M_T}} P_{RNN}(w_i|h_i, \hat{f}_{M_T^{k(n)},d'}), \tag{6.9}$$

where the topic posterior $\hat{f}_{\mathcal{M}_T^k,d'}$ can be obtained by maximising $P(d'|\mathcal{M}_T^k)$. However, directly computing Equation 6.9 is not practical as it requires training multiple RNNLMs for varying numbers of topics. To address this issue, the MAP estimate $\hat{\mathcal{M}}_T = \arg\max P(\mathcal{D}|\mathcal{M}_T^k)$ is used as an approximation. The open-source toolkit[2] for HDP based on MCMC sampling is used in this work. The topic posterior probabilities $P(z_k|d')$ on test document d' are computed as in Equation 6.8.

[2] http://www.cs.princeton.edu/ chongw/resource.html

6.4 Experiments

6.4.1 Experimental Setup

An archive of multi-genre broadcast television shows was supplied by the British Broadcasting Corporation (BBC) and used for these experiments. A total of seven weeks of BBC broadcasts with original subtitles were available, which gave about 1000 hours of training data after suitable processing and alignment. A carefully transcribed test set containing 16.8 hours of data from 40 shows broadcast from one week was used.

A baseline acoustic model was built using standard PLP cepstral and differentials transformed, with HLDA and modelled with decision tree clustered cross-word triphones, followed by MPE training. An improved Tandem model used 26 additional features generated by a deep neural network (DNN) with a bottleneck [85] layer. Both a speaker independent version of this system (Tandem-MPE) and one with CMLLR-based adaptive training (Tandem-SAT) were used. The hypotheses from the Tandem-MPE model were used as adaptation supervision. Details of the construction of Tandem acoustic models can be found in [129].

The baseline 4-gram (4g) language model was trained on about 1 billion words of text collected from US English broadcast news and the 11 million words of BBC acoustic model transcription with slight prunning, which includes 145M 3-gram and 164M 4-gram entries. These 11 million words are from the transcription of 6 weeks BBC data, consisting of 2231 shows. A 64K word list was used for decoding and language model construction. The RNNLM was trained on the 11M words using a 46K word input shortlist and 35K output shortlist. The 2231 BBC shows are labelled with 8 different genres (advice, children, comedy, competition, documentary, drama, event and news).

Genre	Train		Test	
	#token	#show	#token	#show
advice	1.8M	269	24.4K	3
children	1.0M	418	20.8K	7
comedy	0.5M	154	27.2K	5
competition	1.6M	271	25.8K	6
documentary	1.6M	302	57.8K	6
drama	0.8M	149	20.3K	3
events	1.2M	180	28.7K	5
news	3.1M	488	22.2K	5
Total	11.5M	2231	227.1K	40

Table 6.1 Statistics of the BBC training and test data.

Table 6.1 gives the statistics of the 11M BBC data. The average sentence length (with sentence start and end) on the subtitle training set and the test set with manual segmentation are 19.3 and 9.7 respectively and the OOV rate is 1.39%. The corpus is shuffled at the sentence level for RNNLM training. Stop words are filtered out for training of topic

representations [3] as they are usually refer to the most common, short function words in a language, such as “a”, “the”, “is”, and “on” in English. For training of genre dependent RNNLMs, a genre independent model is first trained on all 11M data, then followed by fine-tuning on genre-specific data or the use of a genre input code. To allow the use of show-level topic adaptation, RNNLMs were trained from scratch with the topic representation as an additional input.

The RNNLMs had a single hidden layer with 512 hidden nodes and were trained on a GPU with a 128 bunch size [39]. RNNLMs were used in lattice rescoring with a 4-gram approximation as described in Chapter 5. All word error rate (WER) numbers are obtained using confusion network (CN) decoding [145]. For all results presented in this chapter, matched pairs sentence-segment word error (MAPSSWE) based statistical significance test was performed at a significance level of $\alpha = 0.05$.

6.4.2 Results for RNNLMs trained on 11M words

Table 6.2 gives the PPL and WERs for genre dependent RNNLMs. Tandem-SAT system was used as acoustic model. From the results, the use of genre independent RNNLMs gives a significant WER reduction of 0.7% absolute. Genre dependent RNNLMs trained using both fine-tuning and genre-codes both gave small statistically significant WER reductions. The use of a genre-code is preferred since only one RNNLM needs to be stored.

LM	PPL		WER
	RNN	+4glm	
4glm	-	123.4	32.07
RNN	152.5	113.5	31.38
+fine-tuning	148.7	110.4	31.29
+genre-code	144.2	109.3	31.24

Table 6.2 PPL and WER results for genre dependent RNNLMs on 1 week BBC test data. The genre dependent RNNLMs were constructed based on the well-trained genre independent RNNLM. Tandem-SAT system was used as acoustic model.

In the next experiment, RNNLMs trained with show-level topic representations were evaluated. In [151], each sentence was viewed as a document in the training of LDA, and a marginal (0.1%) performance gain was reported on a system using an MPE-trained acoustic model. In this work, each show is processed as a document for robust topic representation. The test-set topic representation is found from the recognition hypotheses using the 4-gram LM after CN decoding. For comparison purposes the reference transcription is also used. For PLSA and LDA, the number of topics is 30 unless otherwise stated.

An initial experiment used the baseline MPE acoustic model. The RNNLM gave 0.7% absolute WER reduction over the 4-gram LM, and the LDA based unsupervised adaptation gave a further 0.4% WER reduction. The experimental results using Tandem-SAT acoustic

[3] Using stop words didn’t affect performance in our experiments.

models are shown in Table 6.3. PLSA and LDA gives comparable PPL and WER results. A 0.2% to 0.3% WER improvement[4] and 8% PPL reduction were achieved. This is consistently better than genre-dependent RNNLMs. It is worth noting that the PLSA and LDA derived from reference (supervised) and hypotheses (unsupervised) gave comparable performances. This indicates that the topic representation inference is quite robust even when the WER is higher than 30%. The number of topics chosen by HDP is 24, giving a slightly poorer PPL and WER than LDA and PLSA. It is maybe related to parameter tuning since the number of topics chosen by HDP was found to be sensitive to initial parameters. Table

Topic M	Sup	PPL		WER
		RNN	+4glm	
-	-	152.5	113.5	31.38
PLSA	hyp	137.8	106.3	31.16
	ref	137.3	105.1	31.08
LDA	hyp	133.7	105.0	31.14
	ref	134.1	104.2	31.07
HDP	hyp	138.9	106.6	31.19
	ref	138.0	105.2	31.10

Table 6.3 PPL and WER results for RNNLMs with various topic representation on 1 week BBC test data. The RNNLMs with topic representation were trained from scratch. The number of topics is set to 30 for all topic models. The hypotheses were obtained using the 4-gram LM.

6.4 gives the PPL and WER results with different numbers of LDA topics derived from the reference. The results show that the performance is fairly insensitive to the number of topics and 30 gives the best performance in terms of PPL and WER.

Topic Dim	PPL		WER
	RNN	+4glm	
20	138.7	106.4	31.13
24	139.3	105.8	31.16
30	134.1	104.2	31.07
40	137.1	104.3	31.11

Table 6.4 PPL and WER results for RNNLM adaptation with LDA using different numbers of topics on 1 week BBC test data

6.4.3 Results for RNNLM trained with additional Subtitle Data

The baseline RNNLMs with the previous seting were rebuilt using an additional 620M of BBC subtitle data. A 4-gram LM trained on the 620M BBC subtitle data was interpolated

[4]WER improvements are statistically significant.

LM	#hidden node	Topic Model	PPL	WER
4glm	-		103.1	25.61
+RNN	512	-	93.0	25.03
		LDA	85.1	24.71
	1024	LDA	81.0	24.36

Table 6.5 PPL and WER on 1 week BBC test data using RNNLM trained on additional subtitle data.

with the 4-gram LM trained on 1.6 billion words including the 620M BBC subtitle data. More advanced acoustic model techniques were used to improve the baseline system, which was used for the ASRU2015 MGB challenge [242]. RNNLMs were trained on all 630M of text, consisting of the 620M BBC subtitles and the 11M of acoustic model transcription[5]. RNNLMs with 512 and 1024 hidden nodes were used and compared.

Table 6.5 presents the PPL and WER results with the additional 620M words of BBC subtitles. RNNLM with 512 hidden nodes trained on 630M gives a further 0.6% reduction in WER. RNNLMs with LDA topic features provided an additional 0.3% WER reduction[6] and a 8.5% PPL reduction with unsupervised topic adaptation. Furthermore, RNNLM with 1024 hidden nodes, adapted by LDA gives further 0.3% WER improvement. RNNLM gives a total of 1.2% WER reduction over the baseline ASR system.

6.5 Summary

In this chapter, RNNLM adaptation at the genre and show level were compared on a multi-genre broadcast transcription task. A number of adaptation approaches were examined. Simple fine-tuning on genre specific training data and the use of a genre code as an additional input give comparable performances. Continuous vector topic representations including PLSA, LDA and HDP were incorporated into the training of RNNLMs for show-level adaptation, and consistently outperformed genre level adaptation. Significant perplexity and moderate WER reductions were achieved for speech recognition. Furthermore, the use of LDA based topic adaptation was also effective and offered consistent improvement when RNNLMs were trained on a much larger corpus.

[5]The 11M acoustic transcription was placed after 620M subtitle data in the training data.
[6]WER reduction is statistically significant.

Chapter 7

Interpolating RNNLMs and *n*-gram LMs

The characteristics and generalisation patterns of *n*-gram LMs and RNNLMs are expected to be different and possible complementary to each other. In order to take a advantage of the strengths of these different models, *n*-gram LMs and RNNLMs are usually combined using a context independent, fixed weighting based linear interpolation in state-of-the-art ASR systems [153, 154, 217, 32, 38]. The same approach was previously used to combine multiple *n*-gram LMs trained on a diverse collection of data sources. As discussed in Chapter 2.3.3, in order to reduce the mismatch between the interpolated LM and the data of interest, interpolation weights are often tuned by minimizing the perplexity on data from target domain [110, 119, 106, 187, 49]. These interpolation weights indicate the "importance" of individual component LMs for a particular task.

In order to fully exploit the locally varying complementary attributes among component LMs during interpolation, a more general history dependent form of interpolation can be used to combine *n*-gram LMs [141] where the interpolation weights depend on the word history. A similar local variation of probabilistic contribution from *n*-gram LMs and RNNLMs across different contexts during interpolation has been also reported [171]. The perplexity analysis over *n*-gram LMs and RNNLMs in [171] suggests this variation is heavily correlated with the underlying context resolution of component *n*-gram LMs. For example, RNNLMs assign higher probabilities when the *n*-gram LMs' context resolution is significantly reduced via the back-off recursion to a lower order, and conversely when a longer history can be modelled by the *n*-gram LMs without using back-off. Inspired by these findings, a back-off based compact representation of *n*-gram dependent interpolation weights is described in this chapter. This approach allows robust weight parameter estimation on limited data. Experiments are conducted on the three tasks with varying amounts of training data. Small and consistent improvements in both perplexity and WER were obtained using the proposed interpolation approach over the baseline fixed weighting based linear interpolation.

This chapter is organised as follows. Linear interpolation is first briefly reviewed in Chapter 7.1. Back-off based interpolation is investigated in Chapter 7.2 and two methods for back-off based interpolation are proposed and compared. Experiments are conducted in Chapter 7.3 and this chapter is concluded in Chapter 7.4.

7.1 Linear Interpolation

Methods to combine multiple language models had been studied and compared in [22, 141, 101]. These techniques are investigated on *n*-gram LMs and their derivations, such as topic based *n*-gram LM and cached based *n*-gram LM as discussed in Chapter 2.3.6. RNNLMs are inherently different from *n*-gram LMs in terms of their generalisation patterns. For this reason, RNNLMs are usually linearly interpolated with *n*-gram LMs to obtain both a better context coverage and strong generalisation [153, 217, 196, 62, 172, 131]. The interpolated LM probability is given by,

$$P(w_i|w_0^{i-1}) = \frac{1}{Z(h_i)}\Big(\lambda P_{\mathsf{NG}}(w_i|w_0^{i-1}) + (1-\lambda)P_{\mathsf{RN}}(w_i|w_0^{i-1})\Big) \tag{7.1}$$

where $Z(h_i)$ is the normalisation term, $h_i = w_0^{i-1}$ represents the complete history of w_i, and λ is the global weight of the *n*-gram LM distribution $P_{\mathsf{NG}}(\cdot)$, which can be optimised using the EM algorithm on a held-out set.

To guarantee the interpolated probability $P(w_i|w_0^{i-1})$ be a valid probability, the sum to one constraint needs to be satisfied,

$$\sum_{w_i \in \mathcal{V}} P(w_i|w_0^{i-1}) = 1 \tag{7.2}$$

where V is the vocabulary. For linear interpolation shown in Equation 7.1, the normalisation term can be written as,

$$\begin{aligned} Z(h_i) &= \sum_{w_i \in V} \Big(\lambda P_{\mathsf{NG}}(w_i|w_0^{i-1}) + (1-\lambda)P_{\mathsf{RN}}(w_i|w_0^{i-1})\Big) \\ &= \lambda + (1-\lambda) = 1 \end{aligned} \tag{7.3}$$

The normalisation term $Z(h_i)$ is always 1 in linear interpolation. Hence it provides a simple way to combine two language models.

7.2 Back-off Based LM Interpolation

7.2.1 Generalised LM Interpolation using Weight Clustering

As discussed in Chapter 2.3.3, in order to fully exploit the complementary attributes among different language model architechtures, a more general form of linear probability interpolation between *n*-gram LMs and RNNLMs based on word and history dependent weights can be considered as below,

$$P(w_i|w_0^{i-1}) = \frac{1}{Z(h_i)}\Big(\lambda^{(\mathsf{NG})}_{(w_i,w_0^{i-1})} P_{\mathsf{NG}}(w_i|w_0^{i-1}) + \lambda^{(\mathsf{RN})}_{(w_i,w_0^{i-1})} P_{\mathsf{RN}}(w_i|w_0^{i-1})\Big) \tag{7.4}$$

where $\lambda^{(\mathsf{NG})}_{(w_i,w_0^{i-1})}$ is the interpolation weight for n-gram LM and $\lambda^{(\mathsf{RN})}_{(w_i,w_0^{i-1})}$ is the interpolation weight for RNNLM given the predicted word w_i and history w_0^{i-1}. This term is computed as,

$$Z(h_i) = \sum_{w' \in V} \Big(\lambda^{(\mathsf{NG})}_{(w',w_0^{i-1})} P_{\mathsf{NG}}(w'|w_0^{i-1}) + \lambda^{(\mathsf{RN})}_{(w',w_0^{i-1})} P_{\mathsf{RN}}(w'|w_0^{i-1}) \Big) \tag{7.5}$$

$Z(h_i)$ is the normalisation term to ensure that the predicted word probability mass is a valid function. This approach requires a large number of interpolation weight parameters to be robustly estimated and therefore leads to a data sparsity issue for limited data, which is similar to simply estimating n-gram LMs. A general solution to handle this problem is to share weights within groups of contexts where the contributions from n-gram LMs and RNNLMs are similar. Using this approach a more compact representation of the n-gram dependent interpolation weights can be derived. The weighting based linear interpolation in Equation 7.1 is thus modified to,

$$P(w_i|w_0^{i-1}) = \frac{1}{Z(h_i)} \Big(\lambda^{(\mathsf{NG})}_{\Phi(w_i,w_0^{i-1})} P_{\mathsf{NG}}(w_i|w_0^{i-1}) + \lambda^{(\mathsf{RN})}_{\Phi(w_i,w_0^{i-1})} P_{\mathsf{RN}}(w_i|w_0^{i-1}) \Big) \tag{7.6}$$

where the n-gram dependent interpolation weights $\lambda^{(NG)}_{\Phi(w_i,w_0^{i-1})}$ and $\lambda^{(RN)}_{\Phi(w_i,w_0^{i-1})}$ are positive values and shared using an n-gram clustering function $\Phi(\cdot)$. A normalisation term $Z(h_i)$ is also required to ensure the interpolated LM probabilities are valid. This normalisation term is computationally heavy as it no longer always equal to 1 as linear interpolation in Equation 7.3.

The above form of interpolation based on n-gram weight classing is illustrated in Figure 7.1. Usually, the interpolation weights of n-gram LM and RNNLM satisfy,

$$\lambda^{(\mathsf{NG})}_{\Phi(w_i,w_0^{i-1})} + \lambda^{(\mathsf{RN})}_{\Phi(w_i,w_0^{i-1})} = 1. \tag{7.7}$$

By definition, the standard fixed weight based linear interpolation in Equation 7.1 is subsumed by the more general form of linear interpolation in Equation 7.6, and is equivalent to assigning all contexts to a single class and fixed interpolation weights are used.

7.2.2 Interpolation using Back-off for Weight Clustering

A central part of the n-gram class dependent interpolation approach given in Equation 7.6 is to derive an appropriate form of context class mapping $\Phi(\cdot)$. For interpolation between back-off n-gram LMs and RNNLMs, a suitable weight classing scheme is expected to reflect the variation of the probabilistic contribution from these two component LMs. In previous research it was found that this variation was heavily correlated with the n-gram LM's underlying context resolution [171]. This is represented by the highest available n-gram order obtained through the back-off recursion in n-gram LM in Equation 2.27.

The back-off scheme in n-gram LM was first discussed in Chapter 2.3.2 and is recalled here. An example back-off recursion for a 3-gram LM is illustrated in Figure 7.2. When

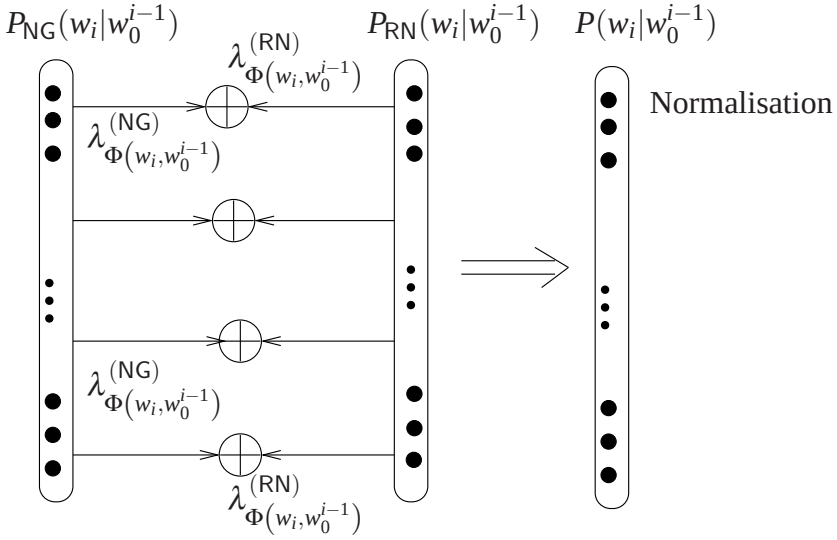


Fig. 7.1 *n-gram dependent interpolation of n-gram LM and RNNLM.*

the trigram probability $P(w_i|w_{i-1},w_{i-2})$ is estimated directly in the 3-gram LM, there is no back-off and the back-off level is 3. When the $P(w_i|w_{i-1},w_{i-2})$ does not exist and its back-off 2-gram LM probability is used instead, the back-off level is 2. Similarly, when it backs off to unigram, then the back-off level is 1.

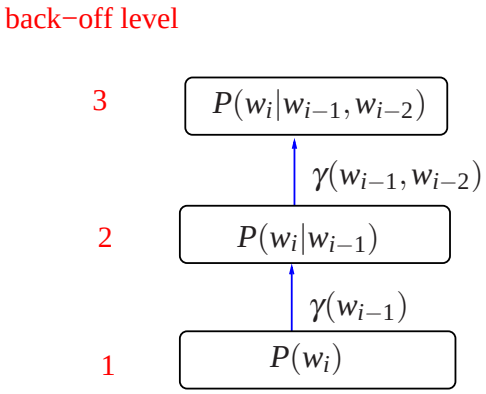


Fig. 7.2 *An illustration of back-off scheme in a trigram LM.*

This correlation in perplexity is presented again in this chapter based on the Penn Tree-Bank (PTB) corpus as in [171]. A detailed breakdown of the perplexity performance of the baseline 5-gram LM, RNNLM and their linear interpolation over different *n*-gram context groups of varying back-off orders on the PTB test data is shown in Table 7.1. The 5-gram LM outperformed the RNNLM in terms of the overall perplexity. As expected, significant perplexity reduction can be obtained using standard optimised linear interpolation as shown in 7.1 (3rd line in Table 7.1). A large variation in the 5-gram LM's contribution characterised by the rank ordering in perplexity against the RNNLM over different back-off orders is also clearly shown in Table 7.1. This is due to the fact that *n*-gram LMs and RNNLMs

employ inherently different mechanisms to acquire generalisation. *n*-gram LMs are more powerful in predicting the probabilities of frequently occurring *n*-grams using higher order context modelling, while RNNLMs' strength lies in their ability to predict rare events.

Table 7.1 Perplexity performance of baseline 5-gram LM, RNNLM and their linear interpolation over varying back-off *n*-gram orders on PTB test set.

LM	*n*-gram LM back-off level					Overall
	1g	2g	3g	4g	5g	
#words	15594	33646	19655	9502	4033	82430
5glm	9157.3	198.0	26.4	8.3	2.5	141.5
RNN	4633.7	183.4	38.7	17.9	6.0	150.8
5glm+RNN	4697.6	161.6	26.9	9.4	3.0	118.3

The above analysis suggests that the back-off order can provide an indication of *n*-gram level varying probabilistic contribution. The interpolation weights can be clustered into a small number of classes based on their back-off orders. The weight parameters can be robustly estimated even on a small amount of held-out data. The associated interpolation weight classing is thus computed as,

$$\begin{aligned}\Gamma(w_i, w_0^{i-1}) &= \Gamma_{\mathsf{NG}}\left(w_i, w_{i-n+1}^{i-1}\right) \\ &= \begin{cases} n & \text{if } <w_i, w_{i-n+1}^{i-1}> \in G_{\mathsf{NG}} \\ \Gamma_{\mathsf{NG}}\left(w_i, w_{i-n+2}^{i-1}\right) & \text{otherwise} \end{cases}\end{aligned} \tag{7.8}$$

where $G_{\mathsf{NG}} = \{..., <w', h'>, ...\}$ contains all the unique observed *n*-gram contexts that the *n*-gram LM $P_{\mathsf{NG}}(\cdot)$ models. $\Gamma(w_i, w_0^{i-1})$ can be viewed as the back-off order in the *n*-gram LM for word w_i with history w_0^{i-1}. However, the optimisation of interpolation weights is not easy due to the normalisation term $Z(h_i)$ in Equation 7.5. Stochastic gradient descent is applied for the optimisation on a held-out set. The interpolation weight for *n*-gram LM and RNNLM $\lambda^{(\mathsf{NG})}_{\Gamma\left(w_i, w_0^{i-1}\right)}$ and $\lambda^{(\mathsf{RN})}_{\Gamma\left(w_i, w_0^{i-1}\right)}$ can be any positive value in theory. Normally the following condition is retained for each back-off level during optimisation.

$$\lambda^{(\mathsf{NG})}_{\Gamma\left(w_i, w_0^{i-1}\right)} + \lambda^{(\mathsf{RN})}_{\Gamma\left(w_i, w_0^{i-1}\right)} = 1 \tag{7.9}$$

This form of interpolation will be denoted as *n*-gram LM $\oplus$ RNNLM and the interpolated probability can be written as,

$$P^{\oplus}(w_i|w_0^{i-1}) = \frac{1}{Z(h_i)}\left(\lambda^{(\mathsf{NG})}_{\Gamma\left(w_i, w_0^{i-1}\right)} P_{\mathsf{NG}}(w_i|w_0^{i-1}) + \lambda^{(\mathsf{RN})}_{\Gamma\left(w_i, w_0^{i-1}\right)} P_{\mathsf{RN}}(w_i|w_0^{i-1})\right) \tag{7.10}$$

7.2.3 Back-off based Interpolation with Rescaling

The *n*-gram class dependent interpolation approach given in Equation 7.10 requires a normalisation term $Z(h_i)$ to be computed for each distinct history over multiple weight classes.

As such term is also dependent on the interpolation weights, a direct optimisation of the weight parameters by maximising the interpolated LM probabilities in Equation 7.10 is a non-trivial problem. Computationally expensive numerical optimisation methods are required.

In order to improve efficiency, an alternative novel form of *n*-gram class dependent interpolation between back-off LMs and RNNLMs is considered. This is given by,

$$P^{\otimes}(w_i|w_0^{i-1}) = \lambda_{\Gamma(w_i,w_0^{i-1})} P_{\mathsf{NG}}(w_i|w_0^{i-1}) + \Big(1-\lambda_{\Gamma(w_i,w_0^{i-1})}\Big)\beta_{\Gamma(w_i,w_0^{i-1})}(w_0^{i-1}) P_{\mathsf{RN}}(w_i|w_0^{i-1})$$

where the *n*-gram context class and history dependent normalisation term $\beta_{\Gamma(w_i,w_0^{i-1})}(w_0^{i-1})$ is independent of interpolation weight parameters and computed as below,

$$\beta_k(w_0^{i-1}) = \frac{\sum_{w' \in \psi^{(k)}_{w_0^{i-1}}} P_{\mathsf{NG}}(w'|w_0^{i-1})}{\sum_{w' \in \psi^{(k)}_{w_0^{i-1}}} P_{\mathsf{RN}}(w'|w_0^{i-1})} \tag{7.11}$$

where $k = \Gamma(w_i, w_0^{i-1})$, and $\psi^{(k)}_{w_0^{i-1}}$ is the set of word w' whose back-off level equal to k, given history w_0^{i-1}.

$$\psi^{(k)}_{w_0^{i-1}} = \{w' : \Gamma(w', w_0^{i-1}) = k\} \tag{7.12}$$

The whole vocabulary $\mathcal{V}$ can be written as,

$$\mathcal{V} = \bigcup_{k=1}^{K} \psi^{(k)}_{w_0^{i-1}} \tag{7.13}$$

where K is the highest *n*-gram level.

Recalling the general form of interpolation in Equation 7.6, Equation 7.11 is a specific case under the following conditions,

$$\begin{aligned} \lambda^{(\mathsf{NG})}_{\Gamma(w_i,w_0^{i-1})} &= \lambda_{\Gamma(w_i,w_0^{i-1})} \\ \lambda^{(\mathsf{RN})}_{\Gamma(w_i,w_0^{i-1})} &= \Big(1-\lambda_{\Gamma(w_i,w_0^{i-1})}\Big)\beta_{\Gamma(w_i,w_0^{i-1}),w_0^{i-1}} \\ Z(h_i) &= 1 \end{aligned} \tag{7.14}$$

As the above normalisation term is no longer dependent on the interpolation weights, weight parameters associated with different classes can be optimised independently of each other using the conventional EM algorithm on held-out data. During evaluation, this normalisation term can be computed for each pairing of the underlying weight class and history only once and cached for efficiency. In common with the interpolation approach given in Equation 7.6, the form of interpolation in Equation 7.11 also requires a suitable interpolation weight class assignment among different *n*-gram contexts. The back-off based interpolation weight classing given in Equation 7.8 is used.

The resulting back-off based interpolation given in Equation 7.11 retains the probability mass of all *n*-grams sharing a common history and the same back-off based weight class. This probability mass is then be redistributed using the RNNLM distribution during interpolation. In this process, potential bias to the *n*-gram LM distribution may be introduced in the final interpolated LM probabilities. In order to address this issue, it is possible to further improve generalisation performance by combining the interpolated LM probabilities obtained using Equation 7.11 with RNNLMs using the conventional fixed weighting based interpolation.

$$P(w_i|w_0^{i-1}) \;=\; \lambda P^{\otimes}(w_i|w_0^{i-1}) + (1-\lambda)P_{\mathsf{RN}}(w_i|w_0^{i-1}) \tag{7.15}$$

The back-off class dependent interpolation weights $\left\{\lambda_{\Gamma(w_i,w_0^{i-1})}\right\}$ and the top level linear interpolation weight λ can be optimised iteratively using the EM algorithm on a held-out set. This form of interpolation will be denoted as (*n*-gram LM $\otimes$ RNNLM) + RNNLM.

7.3 Experiments

Experiments were conducted on three tasks with different amounts of training data to show the effect of back-off based interpolation. These three corpora are refered as Penn TreeBank (PTB), Babel and Multi-Genra Broadcase (MGB) data. The data in Babel project was used in Chapter 5.5.3 but with different languages and the MGB data used in Chapter 6.4.3. The statistics of the three corpora are given in the following table.

Table 7.2 Statistics of the three corpora used for experiments

Corpus	#Voc	Train	Eval
PTB	10k	860k	79k
Babel	24k	290k	52k
MGB	64k	650M	180k

7.3.1 Experiment on Penn TreeBank Corpus

First, the Penn TreeBank (PTB) corpus was initially used to validate the previous findings reported in [171]. The 860k word PTB training data and a 10k vocabulary were used. A development data set of 70k words was used for parameter tuning. A separate 79k word test test was used for performance evaluation. The perplexity (PPL) results of the 5-gram LM and RNNLM are shown in Table 7.3. The PPL scores of the two LMs over different context groups associated with varying back-off *n*-gram orders are shown in the first two rows. These results were previously presented and discussed in Table 7.1. The third line (5G+RNN) shows the PPL score breakdown of the final linear interpolated LM. The linear interpolation weight λ was perplexity optimised on the development set. According to these results, the conventional form of linear interpolation gave good generalisation performance on each back-off order context group via a simple probability averaging. The overall PPL

was reduced from 141.5 to 118.3. In particular, this standard fixed weighting based interpolated LM gave significant improvements in PPL for the group of contexts where the 5-gram LM backed off to 1-gram.

Table 7.3 PPL results on test set of PTB corpus.

LM	*n*-gram LM back-off level					Overall
	1	2	3	4	5	
#words	15594	33646	19655	9502	4033	82430
5glm	9157.3	198.0	26.4	8.3	2.5	141.5
RNN	4633.7	183.4	38.7	17.9	6.0	150.8
5glm+RNN	4697.6	161.6	26.9	9.4	3.0	118.3
5glm$\oplus$RNN	4568.1	163.0	27.4	9.1	2.9	117.8
5glm$\otimes$RNN	5472.1	170.0	24.3	7.9	2.4	117.6
(RNN$\otimes$5glm)+RNN	5230.7	167.8	24.7	8.1	2.5	117.0

The fourth line (5glm$\oplus$RNN) gives the results of the first back-off based interpolation method introduced in Chapter 7.2.2, where the interpolation weights were optimised with stochastic gradient descent. It gave a slight overall PPL improvement. The fifth line (5glm$\otimes$RNN) presents the results of the back-off based interpolation approach of Chapter 7.2.3. As discussed, this form of back-off based interpolation retains the *n*-gram LM's probability mass of all *n*-grams sharing a common history and the same back-off order based weight class, and re-distributes it using the RNNLM distribution during interpolation. It can be seen from Table 7.3 that the PPL score was improved on each back-off level compared to the baseline 5-gram LM. The interpolation weight λ_n on each back-off level was efficiently optimised independently via the EM algorithm on the development data. The optimal interpolation weights $\{\lambda_n\}$ for the 5-gram LM were $\{0.25, 0.4, 0.5, 0.55, 0.55\}$ for varying back-off levels from 1 to 5. As expected, a general trend could be found that the *n*-gram weight increases with the back-off order. The back-off based interpolated LM probabilities could be further linearly interpolated with the RNNLM (with a weighting 0.9:0.1) using Equation 7.15. This gave further small improvements in perplexity.

7.3.2 Experiments on Babel Corpus

The next experiment was conducted on the BABEL corpus (i.e. IARPA-babel202b-v1.0d) and used Full Language Pack (FLP) of the Swahili language. A 3-gram LM (3glm) with slight pruning and RNNLM were both trained on 290K words of text data[1]. The test set includes 52K words. The vocabulary size is 24K. All vocabulary words were used in RNNLM input and output word listst during training. A total of 200 hidden units were used. RNNLMs were trained on GPU as described in [40]. The PPL and WER results are shown in Table 7.4. A pattern similar to that observed on the PTB task in Table 7.3 was found. Standard linear interpolation reduced the overall PPL score by 7% relative compared with the 3-gram LM. A detailed analysis on each back-off level showed that linear

[1] A 4-gram LM gave no further improvements of ASR performance given the small amount of training data

interpolation improved the PPL score by 25% relative on the words where the 3-gram LM backs off to 1-gram, while no improvements were obtained for the other two back off levels. The back-off based interpolation by simply clustering (3glm$\oplus$RNN) provided slight PPL reduction and obtained the same WER as linear interpolation. The back-off based interpolation (3glm$\otimes$RNN) reduced the PPL consistently on each back-off level compared with the 3-gram LM. A small overall PPL reduction was also obtained over the conventional fixed weight based linear interpolation. The optimal interpolation weights assigned to the 3-gram LM were (0.25, 0.6, 0.65) for the back-off levels from 1-gram to 3-gram.

Table 7.4 PPL and WER results on Swahili for Babel

LM	PPL				WER
	n-gram LM back-off level			Overall	
	1	2	3		
#words	19687	28355	7156	55198	
3glm	3510.4	107.1	19.0	297.2	47.3
RNN	2387.3	145.6	29.9	321.6	-
3glm+RNNLM	2618.3	107.9	21.5	273.0	46.8
3glm$\oplus$RNN	2602.6	107.6	21.6	272.2	46.8
3glm$\otimes$RNN	2933.9	102.0	18.7	271.3	46.9
(3glm$\otimes$RNN)+RNN	2850.4	103.0	19.2	270.7	46.7

ASR experiments were then conducted on the same BABEL task. The acoustic models were trained on 46 hours of speech. Tandem and hybrid DNN systems were trained separately. A frame level joint decoding was then applied to combine the acoustic scores of the two systems [233]. The baseline 3-gram LM was used in the first decoding stage for lattice generation. N-best (N=50) rescoring was then applied using the interpolation between the RNNLM and 3-gram LM. The word error rate (WER) results are shown in Table 7.4. The baseline 4-gram gave a WER score of 47.3%. Standard linear interpolation gave an absolute 0.5% WER reduction. the back-off based interpolation gave a comparable WER score of 46.9%. A further linear interpolation using Equation 7.15 between the back-off based interpolated LM and the RNNLM (with a weighting 0.9:0.1) gave the lowest WER of 46.7% in the table. A statistical significance test was carried out and it indicated that the improvement from back-off based interpolation was statistically insignificant.

7.3.3 Experiments on MGB Corpus

The previous experiments were conducted on a relatively small amount of training data. In the next experiment a much larger training set based on the BBC Multi-Genre Broadcast (MGB) challenge task was used[2]. 650M words of text data were used in the baseline 4-gram LM (4glm) and RNNLM training. The hybrid DNN acoustic model was trained on 700 hours of data. A 64K vocabulary was used. A total of 500 hidden nodes were used in

[2]The detail of MGB challenge could be found from http://www.mgb-challenge.org/

the RNNLM. A 46K input shortlist and 40K output shortlist were used in RNNLM training. The results are shown in Table 7.5. The complementary attributes of 4-gram LM (4glm) and the RNNLM on each back-off level were consistent with the previous two tasks. There was only 3.6% of all *n*-gram requests that back off to 1-gram due to the large amount of training data being used and low pruning threshold. In common with the previous two tasks, RNNLM was found to perform better than 4-gram LM when the latter backs off to 1-gram or 2-gram probabilities, while vice versa when it retained a 3-gram or 4-gram modelling resolution. The baseline linear interpolation gave a significant overall reduction in perplexity. On each back-off level, the reduction in perplexity increases when the back-off level decreases. Again the back-off based interpolation with simply clustering (4glm$\oplus$RNN) gave slight PPL improvement and the same WER compared to linear interpolation. The back-off based interpolation with rescaling (4glm$\otimes$RNN) slightly outperformed the conventional linear interpolation. In terms of WER results, the linear interpolation reduced the WER by 0.7% absolutely. 4glm$\otimes$RNN and 4glm$\otimes$RNN+RNN gave a small further reduction of of 0.1% absolute and gave an overall improvement 0.8% absolute over the baseline 4-gram LM. Again, the improvement from back-off based interpolation is statistically insignificant according to the result of a statistical significance test.

Table 7.5 PPL and WER results on MGB task

LM	PPL					WER
	n-gram LM back-off level				Overall	
	1	2	3	4		
#words	7362	60578	78528	56291	202759	
4glm	18731.7	733.2	76.0	11.7	108.7	26.2
RNN	5868.1	564.3	79.0	21.8	116.2	-
4glm+RNN	6782.0	560.3	68.2	13.4	96.3	25.5
4glm$\oplus$RNN	6440.7	563.0	68.7	12.7	95.1	25.5
4glm$\otimes$RNN	7145.7	593.5	70.0	11.5	94.9	25.4
(4glm$\otimes$RNN)+RNN	6800.4	584.2	69.5	11.8	94.7	25.4

7.4 Conclusion and Discussion

In order to exploit the complementary features among *n*-gram LMs and RNNLMs, a standard form of linear interpolation based on fixed weights is widely used. Motivated by their inherently different generalisation patterns that are correlated with the variation of the underlying *n*-gram LM context resolution, a novel back-off based compact representation of *n*-gram dependent interpolation weights is proposed. The proposed technique allows the interpolation weights shared at each back-off level to be estimated both efficiently and robustly. Experimental results on three tasks of varying amounts of training data show that the proposed back-off based linear interpolation between *n*-gram LMs and RNNLMs provided a simple but powerful way to combine them. Small but consistent improvements in terms of

both perplexity and WER reductions were obtained over the conventional fixed weighting based linear interpolation.

Chapter 8

Experiments on Meeting Transcription

In this chapter, the techniques discussed in the previous chapters will be applied to a meeting transcription task. Meeting transcription is a very useful, and highly challenging, task. Many researches have examined a range of approaches for automatically transcribing meeting data [184, 89, 20, 90]. These have normally been applied to standard corpora, such as those used for NIST evaluations [66, 67]. The well-known AMI corpus and related public meeting data are used in this work to allow a contrast with the previous published results. In addition to this publicly available meeting data, the Speech Group at Toshiba Research Europe Ltd, Cambridge Research Laboratory, undertook the recording of meetings, related to speech recognition and synthesis projects, over a number of months. This corpus will be referred to as the Toshiba Technical Meeting (TTM) data. The performance on TTM data is expected to reflect the performance on real-life meeting with mismatched acoustic condition. The ASR system is constructed with HTK [252] on the public meeting corpora. Various language models including *n*-gram LMs, feedforward and recurrent NNLMs are evaluated and compared.

8.1 Data Description

Two corpora are used in this chapter. The first is from the AMI project [27]. This data is used for training acoustic and language models, and as stated to allow comparason with existing systems. The second is the TTM data, which is used as evaluation test set to reflect the performance on a less mismatched meeting data. For both corpora only the multiple distant microphone (MDM) data is used. Beamforming is performed using the *BeamformIt* tool [158] to yield a single audio channel[1].

8.1.1 AMI Corpus

The Augmented Multi-party Interaction (AMI) corpus [27] was collected for research and development of technology that may help groups interact better. As part of this corpus,

[1]Currently there is no Wiener filtering in the front-end processing, as used for example in [89], which should yield performance gains.

speech data was collected in close-talking and far-field microphones, and high quality transcriptions generated. This data was collected in a scenario set-up where 4 people were allocated roles and asked to discuss the design of a remote control unit [27]. In this thesis only the far-field microphones, multiple distant microphone data (MDM) was used as this is felt to be the scenario for meeting transcription application. Additionally, the overlapping speech data was removed from both training and test data. This yielded about 59 hours of data. In addition to the AMI corpus, 52 hours from the ICSI corpus [108] and 10 hours from the NIST corpus were also used [73] for training. ICSI meeting data was recorded in the conference room in ICSI. These meetings were recorded as well as the group meeting, discussing research at ICSI. NIST also provides a pilot meeting data corpus.

Table 8.1 Summary of the AMI Test Data

Test-Set	Meeting id	# Speakers	Hours		
			Duration	Ref. Seg	Auto. Seg
Dev	IS09	4	1.82	1.28	1.49
	ES09	4	1.92	1.37	1.62
Eval	IS08	4	1.59	1.12	1.25
	ES08	4	2.22	1.52	1.77
	Totals		7.55	5.29	6.14

Table 8.2 Summary of the Meeting Training Data

Corpus	# Session	# Speaker	Ref. Seg (Hours)
AMI	152	175	59
ICSI	52	75	52
NIST	19	51	10
Totals	223	301	121

Four meetings were held back from the AMI data to provide an AMI dev and eval set, each contains two meetings, with 4 speakers per meeting. The detailed analysis of this data is given in Table 8.1. The two meetings labelled as IS09 and ES09 were chosen as dev test set; IS08 and ES08 were used as eval test set. As overlapping speech was not evaluated this yielded a total test set size of about 5.29 hours, as shown in the *Ref. Seg* column in the above table. The quantity of automatic segmented data is larger than that in the reference as it includes overlapping speech which is ignored for scoring.

The total available data for training, after removing the 4 test meetings, was about 121 hours of data. The breakdown of the training data can be seen in Table 8.2. This is the same configuration, and held-out test sets, as used in [20].

8.1.2 Toshiba Technical Meeting Data

The second corpus was collected at Toshiba Research Europe Ltd's Cambridge Lab. The corpus was collected in a meeting room (shown in Figure 8.1) with between 6 and 9 participating in each meeting. The data was recorded using a microphone array. A single microphone array was placed in the centre of a meeting room[2]. This limited the nature of the data that can be collected: no close-talking microphone data is available; individuals were able to sit where they wanted; and move to give presentations or have "side" conversations. The Toshiba ASR and TTS technical meetings were recorded, these involved discussion of the on-going research projects and future plans. Compared to the AMI corpus, the TTM data has a greater distance from the microphone to the speaker and a higher level of noise. The level of background noise is also much higher in this meeting compared to both the training data and the AMI testsets. A subjective estimate of SNR of the meeting is around 0 to 5dB but it varies by speaker. These differences will be reflected in a higher baseline WER than for a typical meeting from the AMI corpus. Additionally it was agreed that this data would never be made publicly available or used to assess the performance of individuals.

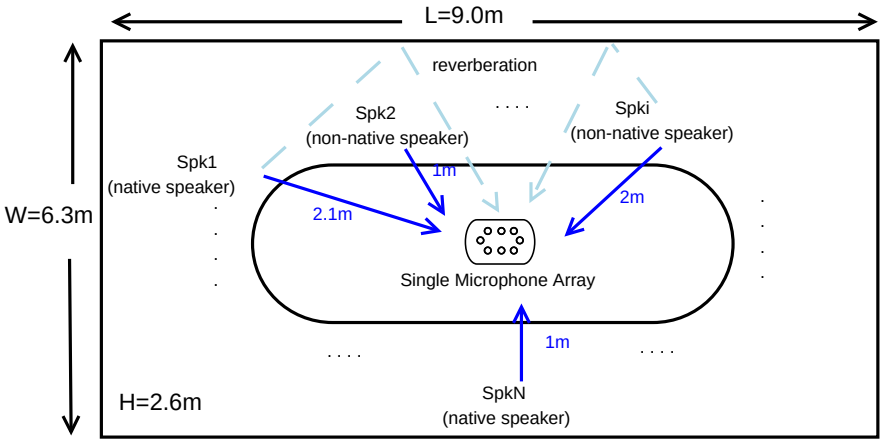


Fig. 8.1 *Toshiba Technical Meeting Recording Configuration*

A sequence of TTM meetings were recorded over two years. 158 shows are collected, which are 179 hours in length. All meetings were truly held in Toshiba, including 15 shows (20 hours) for ASR project, 61 shows (57 hours) for TTS project, 33 shows (44 hours) about discussion on acoustic modelling, 35 shows from the group meeting (41 hours) and meetings with outside parties, etc. Neither speaker nor reference information is available. This corpus was explored for long-term adaptation as described in [38].

An initial group of seven shows (8.88 hours) were selected as the TTM testset to evaluate performance. Table 8.3 shows the details of these seven meetings[3]. Note that overlapping speech was again removed from the reference.

[2]Multiple microphone arrays could address some of the issues observed with distance from the microphone.

[3]Meeting T0004 had a large amount of silence at the end of the recording (0.63 hours) as the recording equipment was not turned off. This silence was correctly detected by the automatic segmenter. The statistics in table 8.3 are after this data has been ignored.

Table 8.3 Summary of the Toshiba Technical Meeting (TTM) Data

Genre	Meeting	# Speakers	Hours		
			Duration	Ref. Seg	Auto. Seg
ASR	A0001	7	1.83	1.47	1.50
	A0002	7	2.27	2.10	1.72
	A0006	6	1.67	1.59	1.30
TTS	T0001	9	1.08	0.99	0.82
	T0002	8	1.23	0.99	0.91
	T0003	9	1.19	1.01	0.88
	T0004	9	2.11	2.00	1.74
		Totals	11.38	10.16	8.88

Comparing the statistics in Table 8.3 and Table 8.1 indicates that the performance of the automatic segmenter was different on the TTM data than the AMI data, with significant amounts of speech data being missed. This may partly be explained by the larger room and additional noise from a fan in the meeting room in the TTM data.

Manual transcriptions were provided as reference for evaluation. As this data was transcribed early in the collection, it is related to meeting during the intial phase of data collection. To assess the quality of the professional transcriptions, the first recorded meeting (denoted as A0001) was also transcribed by people who were present at the meeting and familiar with the attendees and their accents. This will be referred to as the "gold-standard". Table 8.4 shows both the microphone distance and word error rate of the professional (manual) transcriptions, compared to the "gold-standard", and performance of an initial ASR system. Speakers C, D and E are native UK English speakers, while speakers A and B are non-native. The professional transcribers sometimes chose an incorrect word sequence, though the transcriptions were phonetically similar to the correct word sequence. The overall WER is 10.6%, which indicates the TTM data is a highly challenging task. The transcribers had difficulty with non-native speakers (A and B), Japanese and Chinese names, technical jargon and abbreviations. These are not issues for people familiar with the participants and topics. For example, these are able to generate "gold-standard". However, these "gold-standard" transcriptions are very difficult to obtain as they require transcribers with expert in-domain knowledge of both the meeting topic and participants. For this work the "gold-standard" was used as the reference for meeting A0001, and the manual transcriptions

Table 8.4 Microphone distance and WER results of manual transcription and ASR result compared to "gold-standard" on meeting A0001

Speaker	A	B	C	D	E	Avg
Dist (m)	2.1	2.1	1.2	1.6	1.2	—
Manual	16.4	15.7	8.2	7.6	2.9	10.6
ASR	68.6	66.4	74.9	70.2	55.4	64.8

were used for the other six meetings. The lowest WER speaker for the ASR system was the same as the manual transcribers, a speaker close to the microphone. However, there was no consistent pattern over the other speakers.

8.2 Baseline System

The ASR system was constructed using the HTK v3.5 from Cambridge University [252]. First of all, in order to validate the performance of the HTK toolkit, both Kaldi and HTK toolkits were used to build the Hybrid system using the standard AMI train and test set as Kaldi recipe[4]. The same language model was used for decoding. The experimental results show that they gave comparable performances. More details can be found in Appendix B. All experiments in this chapter are based on the HTK toolkit.

8.2.1 Acoustic Modelling

Two forms of acoustic model were examined for meeting transcription. These will be referred as,

- Tandem System [85]: GMM-based systems using PLP and bottleneck (BN) features ;
- Hybrid System [51]: combining HMMs and DNN posteriors.

All systems were based on state-clustered decision-tree triphone models. The same 6000 distinct states were used for both the GMM and neural network based systems.

A GMM system with PLP feature was built to initialise the baseline system and make use of the publicly available combilex dictionary [185]. The system was built in a similar fashion to that described in [20]. 13-dimensional PLP features were extracted from the data and delta, delta-delta and triples appended. CMN, CVN and Heteroscedastic linear discriminant (HLDA) [127] were then used for feature normalisation and projection. On average each state had 36 Gaussian components. This yielded a 39-dimension feature vector. The minimum phone error (MPE) [178] criterion was used to train these initial acoustic models. Speaker adaptive training (SAT) [3] based on constrained maximum likelihood regression (CMLLR) [68] was also used. Additionally MLLR was used to adapt to the target test speaker for the GMM-based systems. For all systems using speaker adaptation the supervision for the adaptation was obtained from the speaker independent (SI) MPE baseline PLP-based GMM system. Note that adaptation was performed for each speaker per meeting. Thus, though the same speaker appears in multiple meetings this knowledge is not used.

This initial system, was then extended to a Tandem system by appending bottleneck features [85] to the PLP. A deep neural network with four hidden layers (2000 nodes per layer) was constructed. Discriminative pretraining used in [198] was used to train the neural networks. For all systems, 9 frames were spliced together to form the input layer to the MLPs.

[4]The Kaldi recipe for AMI system can be found at https://github.com/kaldi-asr/kaldi/tree/master/egs/ami/s5

For the Tandem feature used in the Tandem system, the bottleneck feature was extracted from the deep neural network using Fbank feature as input feature. The bottleneck feature was appended to the standard PLP feature. The dimension of the bottleneck feature is selected to be 26. A semi-tied covariance (STC) [69] transform was applied to the bottleneck features prior to concatenation with the PLP features. Thus the dimensionality of the Tandem feature here is 65. The Tandem acoustic models were built using the rapid construction approach described in [173]. Again MPE training and SAT models were constructed, with an average of 36 Gaussian components per state.

The second acoustic model was based on a Hybrid system, also known as DNN-HMM systems [51]. The training of neural network in Hybrid system is similar to the training of the BN features in the Tandem system. 9 consecutive frames are concatenated together as input to the neural networks. The DNN is trained in a supervised fashion and discriminatively layer by layer in pretraining, then it is fine-tuned with several epochs until the frame accuracy converges in cross validation set. The alignment for the targets was obtained from the SAT Tandem system. The Tandem feature used in the Tandem system was adopted as input static feature of the deep neural work in the Hybrid system. sequence training [115, 214, 232] was applied for deep neural network (DNN) training.

Joint decoding 2.2.4 is used to combine Tandem and Hybrid systems for better baseline performance. Lattices from joint decoding are then adopted for lattice rescoring using various language models, e.g. RNNLMs [42].

8.2.2 Language Modelling

A variety of sources including the acoustic model transcriptions (from AMI, ICSI, NIST), ISL, Callhome, Switchboard, Gigaword and web data collected by the University of Washington were used for training language model. Language model interpolation weights were tuned on the AMI dev set. A 41K word list was used as the LM vocabulary. 3-gram and 4-gram LMs were trained on a mixture of text corpora. A summary of the LM training corpora can be in in Table 8.5. In total, 2.6G words of language model training data were used. It can be seen that the AMI transcription is the most important corpus, giving a interpolation weight of 0.651. Although the Gigaword corpus contains 1.71G words, it contribute 0.015 in terms of interpolation weight.

Table 8.5 Statistics of the train corpora for the language model. The linear interpolation weights on each corpus were optimised on the AMI dev set.

Text Source	# Words	interpolate weight
Gigaword	1.71G	0.015
University of Washington	910M	0.234
Fisher	21M	0.100
Acoustic Transcription	2.0M	0.651
Callhome+SWB	0.60M	0.001
Total	2.62G	

Table 8.6 gives the out-of-vocabulary (OOV) rates on AMI dev, eval and the TTM test sets. Interestingly the OOV rates for the TTM data were lower than that for the AMI, scenario-based, data.

Table 8.6 Out of Vocabulary (OOV) % for AMI and TTM test data

AMI		TTM
dev	eval	
2.23%	2.17%	1.24%

8.2.3 Baseline Results

Table 8.7 gives the word error results of the baseline with various acoustic models. A 3-gram language model was used for decoding. The results of Tandem and Hybrid systems are shown in the first block, labelled AMI. It can be seen that Hybrid system outperforms Tandem system on the AMI test set and gives the same performance on the TTM test set. Both joint decoding and confusion network decoding (CNC) were applied to combine the Tandem and Hybrid systems. Joint decoding gave slightly better performance than CNC on these three test sets as it combines the acoustic model scores in the early stage. The joint decoding system was served as the baseline system and the lattices generated by joint decoding were used for lattice rescoring to verify the effect of various language models.

Table 8.7 WERs of baseline ASR system on AMI and TTM test sets with a 3-gram language model

AM	AMI		TTM
	dev	eval	
Tandem	32.4	33.5	55.4
Hybrid	31.4	31.9	55.4
Joint decode	30.4	31.0	52.8
CNC	30.5	31.3	53.6

8.3 Experiments with RNNLMs

In addition to the baseline *n*-gram language models, both feed-forward [196] and recurrent neural network (RNN) [153] language models were built. Initially, both neural network based language models (NNLMs) were trained on about 2 million words, the acoustic transcription in Table 8.5. The 41k decoding vocabulary is used as input shortlist and the most frequent 31K words in all train data is chosen as output layer shortlist. An out of shortlist (OOS) symbol was also used on the output layer [196] to represent all OOS words. The

probability of the OOS word is divided by the number of all OOS nodes to get a valid probability over the whole decode vocabulary. Cross entropy is used for the training of RNNLMs in this chapter as the output layer size is not very large.

The feedforward NNLM has two hidden layers with 600 and 400 nodes respectively. The RNNLM was trained using the CUED-RNNLM toolkit [41] with cross entropy criterion. The model was trained on GPU with a bunch size of 64 as discussed in Chapter 4. The size of hidden layer was set to be 256. All the neural network based language models are linearly interpolated with the standard *n*-gram language model and the weight was fixed to be 0.5.

Table 8.8 The perplexity for AMI and TTM test data using various language models. The feedforward and recurrent NNLMs were trained on 2M acoustic transcription.

LM Order	NNLM	AMI		TTM
		dev	eval	
3-gram	—	117.1	110.6	128.8
	Feedforward	98.8	92.8	122.2
	Recurrent	82.8	77.7	112.3
4-gram	—	114.0	108.3	126.7
	Feedforward	90.4	85.4	116.6
	Recurrent	81.5	76.5	110.9

Table 8.8 gives the perplexities of various language models. 3-gram and 4-gram LMs were combined with the feedforward and recurrent NNLMs with linear interpolation. The 4-gram LM gave slight improvement in terms of perplexity compared with the 3-gram LM. It can be seen that both feedforward and recurrent NNLMs helped to reduce the perplexity compared with the baseline *n*-gram LM. RNNLM yielded the lowest perplexity on all three test sets. Comparing the three test sets the TTM data has the highest perplexity. However the increase does not explain the performance degradation in Table 8.7. The impact of the topics associated with the TTM data has not caused a large mismatch with language models.

The word error rate results of these language models are presented in Table 8.9. The WER results are reported on the confusion network decoding [145]. The lattice rescoring using RNNLMs was carried out as described in Chapter 5. The 4-gram LM improved WER by 0.3% to 0.6% on AMI and TTM test sets. Feedforward NNLM gave an additional WER reduction of 0.3% to 0.6%. RNNLM obtained the best performance in terms of WER.

Table 8.10 shows the PPL and WER results of different RNNLM structures and rescoring methods. 256 hidden layer nodes were chosen for both C-RNNLMs and F-RNNLMs. and 200 classes were used for the C-RNNLMs. The WER results of Viterbi decoding is presented in the table. It can be seen that F-RNNLMs gave improved PPL results on AMI dev and eval test sets, as well as better WER results with 100-best rescoring. The lattice rescoring gave comparable WER results with 100-best rescoring. The WERs of RNNLM lattice rescoring can be further reduced to 28.7% and 29.4% on AMI dev and eval sets by applying confusion network decoding on lattice generated from lattice rescoring.

The next experiment was to investigate the LDA based RNNLM adaptation. The experiment were conducted on the 2M AMI acoustic model transcription. According to the

Table 8.9 The WERs for AMI and TTM test data using various language models. The feedforward and recurrent NNLMs were trained on 2M acoustic transcription.

LM Order	NNLM	AMI		TTM
		dev	eval	
3-glm	-	30.4	31.0	52.8
4-gram	-	29.8	30.7	52.5
	Feedforward	29.3	30.1	52.2
	Recurrent	28.7	29.4	51.9

Table 8.10 PPL and WER results of C-RNNLMs (class based RNNLMs) and F-RNNLMs (full output layer RNNLMs) with N-best and lattice rescoring on AMI corpus. The RNNLMs are trained on AMI acoustic model transcription only, which is about 2M and cross entropy is used for RNNLM training. The WER results of Viterbi decoding is presented in the table

LM	rescore	PPL		WER	
		dev	eval	dev	eval
4-gram	-	114.0	108.3	30.4	31.0
+C-RNN	100-best	85.8	81.6	29.0	30.1
+F-RNN	100-best	81.5	76.5	29.5	29.9
	lattice rescore			29.1	30.0

statistics shown in Table 8.2, there are only 223 shows for the whole meeting corpus. The number of meetings is too small to robustly build a topic model. Hence, the sentences from the same speaker in each meeting are regarded as a document. In this way, 1014 documents are collected to estimate the LDA model with 20 topics. In the training of RNNLMs, the posterior probability over topics for each document is computed and used as auxiliary input feature during the LDA based RNNLM training. In test time, the hypothesis from the same speaker in the same meeting is collected as a document and fed to the trained LDA model. The posterior probability vector can be obtained and used for unsupervised adaptation.

Table 8.11 gives the WER results for LDA based RNNLM adaptation. The perplexity results are not given as automatic segmentation was used for evaluation. It is difficult to associate each utterance from automatic segmentation with the reference segmentaion. It can be seen from the table that, for AMI test sets, the WER performance was degraded by 0.1% absolute for both dev and test sets. A degradation of 0.4% was obtained in the TTM test set. The possible reason for the degradation is that the AMI meeting is not as diverse as the MGB data used in Chapter 6. The discussing topics in the meeting data are more homogeneous. And the amount of training data for topic modelling was smaller as well [5]. This phenomenon also needs to be further investigated in the future work. Hence, in the following experiments in this chapter, RNNLMs without adaptation were applied.

[5] using Fisher data don't help to improve performance

Table 8.11 WERs on AMI and TTM test data using 2M transcription using LDA based RNNLM adaptation

Adaptation	AMI		TTM
	dev	eval	
-	28.7	29.4	51.9
LDA	28.8	29.5	52.3

Besides the 2M acoustic model transcription, the additional data for the training of the *n*-gram language model was also used for training RNNLMs. When additional data is used for training, the AMI transcription is always put at the end of the whole train file. Hence, for each epoch, the last seen sentences during training are from AMI transcription, which gave better performance in experiments. Table 8.12 presents the PPL and WER results for RNNLMs with more training data and increased hidden layer size. The first line in Table 8.12 is the performance using RNNLM trained on 2M acoustic model transcription. With additional 24M train words from Fisher, Callhome and Switchboard transcription, the perplexity and word error rate were be reduced slightly. 0.2% WER improvement were obtained on both AMI and TTM test sets. The use of 930M WU data gave good improvement on TTM data. However, in AMI corpus, it helps to eval set, while degrade the WER of dev set. The increase of hidden layer gave consistent PPL and WER improvements. When the layer size size was set to 2048. The WER of TTM test set was 51.1%. The domain adaptation can be carried out by fine-tuning the RNNLMs on the AMI data, which is in-domain data for AMI test sets. The WER on AMI test set can be further improved by fine-tuning, which gives a WER of 27.8% and 28.4% on test and dev set respectively. Hence, the increase of training data and model size gave an additional 0.8% to 1.0% WER reduction compared to the RNNLM trained on 2M transcription.

Table 8.12 PPL and WER results on AMI and TTM test sets with RNNLMs trained on different amounts of training data.

Train Corpus	# train words	#hidden node	PPL			WER		
			AMI		TTM	AMI		TTM
			dev	eval		dev	eval	
AMI	2M	256	81.5	76.5	110.9	28.7	29.4	51.9
+Fisher,Callhome,SWB	24M	512	78.5	74.5	108.1	28.5	29.2	51.7
+WU	930M	512	79.4	75.7	103.4	28.6	29.0	51.3
		1024	74.5	70.8	100.3	28.1	28.9	51.2
		2048	71.8	68.0	98.1	27.9	28.7	51.1
Adapt. (+AMI.finetune)	930M	512	73.1	68.8	101.6	28.2	28.4	51.3
		1024	69.5	65.2	99.5	28.0	28.5	51.2
		2048	67.7	63.0	97.3	27.8	28.4	51.2

The back-off interpolation described in Chapter 7 was not investigated on the meeting data since the improvement over linear interpolation is not statistically significant.

8.4 Summary

In this chapter, RNNLMs were evaluated in the meeting data. The acoustic model was trained on public meeting corpora and two matched test sets from AMI corpus and one mismatched corpus from Toshiba Technical Meeting were used as test sets. The ASR system based on HTK was constructed. Tandem and Hybrid based acoustic models were constructed and then combined using joint decoding in the frame level to get better performance. Various language models including 4-gram LM, feedforward and recurrent NNLMs were evaluated on this system. RNNLMs were trained efficiently on GPU using bunch mode as described in Chapter 4. In test time, lattice rescoring discussed in Chapter 5 was applied and followed by confusion network decoding to get better performance. RNNLMs trained on different amounts of data were investigated and compared. The experimental results show that RNNLMs gave the best performance among these language models. RNNLM trained on 2M acoustic model transcription gave 0.6% to 1.0% WER reduction. The use of additional data allows larger RNNLM to be applied, which further improved the word error rate by 0.7% to 1.0%.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

Language models are crucial components in many applications including speech recognition. *n*-gram LMs have been the dominant language models for many years. However, there are two well-known issues in the standard *n*-gram LMs, which are data sparsity and the *n*-gram assumption. Many sophisticated smoothing techniques [35] have been developed to address the data sparsity issue, and various modifications [130, 125, 187] based on the standard *n*-gram LMs have been proposed to capture longer context for language modelling.

More recently, recurrent neural network was applied to construct word based language model [153] and promising results have been reported on a range of applications and tasks during last several years [154, 61, 216, 55, 123]. Recurrent neural network language models (RNNLMs) provide a good solution for these two problems in the standard *n*-gram LMs. Each word in the input layer of RNNLMs is projected into a low-dimension continuous space, which facilitates the implicit sharing of model parameters. The number of parameters doesn't increase exponentially with the growth of vocabulary size as in the *n*-gram LMs. For the long term history issue in the *n*-gram LMs, the recurrent connection between the hidden layer and input layer is able to model the complete history.

Despite the above advantages of RNNLMs, their long term history characters also bring some problems, especially when they are used in speech recognition. For example, it is difficult to parallel the training using bunch (i.e. minibatch) mode and also hard to apply lattice rescoring as the complete history is required to predict the next word. In this thesis, the application of RNNLMs in speech recognition system is studied in various aspects. The efficient training and inference are investigated in Chapter 4 and the lattice rescoring of RNNLMs is studied in Chapter 5. Researches on adaptation of RNNLMs and interpolation between RNNLMs and standard *n*-gram LMs are discussed in Chapter 6 and 7 respectively.

9.2 Review of Work

In Chapter 4, the efficient RNNLM training and inference are studied. The training of RNNLMs is computationally heavy due to the large output layer and difficulty of paralleli-

sation. The class based output layer was used widely for RNNLM training on CPUs in most previous work. A novel sentence splicing method is proposed in this thesis, which enables RNNLMs to be trained more efficiently with bunch mode. GPU is also used to fully explore its parallelisation power for fast computation. Besides the standard cross entropy based training criterion, two improved training criteria: variance regularisation and noise contrastive estimation, are studied for rapid RNNLM training and inference. Experiments on a conversational telephone speech task show that up to 50 times speedup in terms of training is obtained. The experiments on Google 1 Billion corpus also indicate the scalablity of NCE based RNNLM training.

In Chapter 5, the lattice rescoring of RNNLMs is explored for speech recognition. Due to the long term history, the exact lattice rescoring of RNNLMs is computationally intractable. Previous work used N-best list or prefix tree for RNNLM rescoring, which only rescore the top N hypotheses and can not generate the compact lattice. Approximations are made for RNNLM lattice rescoring to cluster similar histories. n-gram and recurrent vector based clustering are proposed and used as criteria to cluster history and combine paths in lattices. Both of these two approaches are able to generate compact lattices, which can be used in applications including confusion network decoding. Experiments on a conversational telephone speech task and Babel task reveal that performance gain is obtained by applying RNNLM lattice rescoring compared to N-best rescoring.

Chapter 6 describes the work on RNNLM adaptation. Two popular approaches for RNNLM adaptation: model fine-tuning and incorporation of informative feature, are investigated and compared. A range of topic models are used to extract topic representation for efficient adaptation. Experiments on a multi-genre broadcast corpus are conducted to show the performance of RNNLM adaptation.

Chapter 7 investigates the interpolation between RNNLMs and n-gram LMs. Based on an experimental analysis of interpolation between RNNLMs and n-gram LMs, back-off level is used as a feature to cluster and share parameters for interpolation. Two back-off based interpolation algorithms are proposed and investigated. The experiments on three corpora with different amounts of training words show that the back-off based interpolation gives small but consistent performance improvement.

9.3 Future Work

This thesis mainly focuses on the applications of RNNLMs in speech recognition. The efficient training and inference, lattice rescoring, adaptation of RNNLMs and their interpolation with n-gram LMs are studied. However, there are still many areas worthwhile for further investigation. Here a number of suggestions for future directions are listed.

- More efficient training of RNNLMs using multiple GPUs. The work discussed in this thesis focuses on RNNLMs trained on one or two GPUs (pipelined training). For the training of n-gram LMs, it is possible to use many CPUs in parallel to speed up the training. Nowadays, it is not difficult to get multiple GPUs for computation. The training of RNNLMs using multiple GPUs is important and of practical value.

Asychronised training [56] and Hessian free optimization [116] are two promising choices to utilise multiple GPUs for training.

- Modelling context cross sentence. RNNLMs (and LSTM RNNLMs) are able to capture long context. The complete history is used to estimate the word probability. It normally captures history from the beginning of the sentence. However, in many situations, the longer history beyond the current sentence may contain information for prediction, especially in situations where the sentence is quite short, such as voice searching.

- Language model with semantic information. Currently, the statistical language model is applied widely, which is purely data-driven. The semantic information of the sentence is ignored. However, the semantic and grammer knowledge contain information about the structure of sentence, which is also helpful for language understanding. There are also extensive research interests in parsing techniques [228] and word embedding [156]. Hence, these sources of information should be incorporated in language model.

- Multi-language RNNLMs. [182] proposed the training of multi-language RNNLMs. The training corpus consists of sentences from different languages. The input and output layers of RNNLMs are language dependent, while the hidden layer and recurrent connection is shared among different languages. The multi-language RNNLMs outperformed RNNLMs trained on mono language. Under this framework, the multi-language RNNLM allows words from different languages to project into the same low and continuous space. Hence, many aspects can be further explored. such as word embedding for multiple languages and RNNLM adaptation based on multi-language RNNLMs.

References

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*.

[2] Ossama Abdel-Hamid and Hui Jiang. Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code. In *Proc. ICASSP*, pages 7942–7946. IEEE, 2013.

[3] Tasos Anastasakos, John McDonough, Richard Schwartz, and John Makhoul. A compact model for speaker-adaptive training. In *Proc. ICSLP*, volume 2, pages 1137–1140. IEEE, 1996.

[4] Ebru Arisoy, Tara Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Deep neural network language models. In *Proc. NAACL-HLT*, pages 20–28. ACL, 2012.

[5] Ebru Arisoy, Stanley Chen, Bhuvana Ramabhadran, and Abhinav Sethy. Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22(1):184–192, 2014.

[6] Xavier L Aubert. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech & Language*, 16(1):89–114, 2002.

[7] Scott Axelrod, Ramesh Gopinath, and Peder A Olsen. Modeling with a subspace constraint on inverse covariance matrices. In *Proc. ICSA INTERSPEECH*, 2002.

[8] Lalit Bahl, P Brown, P De Souza, and R Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *Proc. ICASSP*, volume 11, pages 49–52. IEEE, 1986.

[9] James K Baker. The Dragon system–an overview. *Acoustics, speech and signal processing, IEEE transactions on*, 23(1):24–29, 1975.

[10] Jerome R Bellegarda. Exploiting latent semantic information in statistical language modeling. *Proc. IEEE*, 88(8):1279–1296, 2000.

[11] Jerome R. Bellegarda. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42:93–108, 2004.

[12] Jerome R Bellegarda, John W Butzberger, Yen-Lu Chow, Noah B Coccaro, and Devang Naik. A novel word clustering algorithm based on latent semantic analysis. In *Proc. ICASSP*, volume 1, pages 172–175. IEEE, 1996.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2): 157–166, 1994.

[14] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137–1155, 2003.

[15] Jeff A Bilmes and Katrin Kirchhoff. Factored language models and generalized parallel backoff. In *Proc. NAACL-HLT*, pages 4–6. ACL, 2003.

[16] Jeff A Bilmes et al. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden Markov models. *International Computer Science Institute*, 4(510):126, 1998.

[17] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008.

[18] David Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[19] Herve A Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer, 1994.

[20] Catherine Breslin, KK Chin, Mark Gales, and Kate Knill. Integrated online speaker clustering and adaptation. In *Proc. ISCA INTERSPEECH*, pages 1085–1088, 2011.

[21] John S Bridle. Alpha-nets: a recurrent 'neural'network architecture with a hidden markov model interpretation. *Speech Communication*, 9(1):83–92, 1990.

[22] Simo Broman and Mikko Kurimo. Methods for combining language models in speech recognition. In *Proc. ISCA INTERSPEECH*, pages 1317–1320, 2005.

[23] Peter Brown, Peter Desouza, Robert Mercer, Vincent Della Pietra, and Jenifer Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4): 467–479, 1992.

[24] Ivan Bulyko, Mari Ostendorf, and Andreas Stolcke. Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures. In *Proc. HLT*, pages 7–9. ACL, 2003.

[25] Lars Bungum and Björn Gambäck. Efficient n-gram language modeling for billion word webcorpora. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, pages 6–12, 2012.

[26] Diamanino Caeiro and Andrej Ljolje. Multiple parallel hidden layers and other improvements to recurrent neural network language modeling. In *Proc. ICASSP*, pages 8426–8429. IEEE, 2013.

[27] Jean Carletta et al. The AMI meeting corpus: A pre-announcement. In *Machine learning for multimodal interaction*, pages 28–39. Springer, 2006.

[28] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell. IEEE, 2016.

[29] Eugene Charniak. Immediate-head parsing for language models. In *Proc. ACL*, pages 124–131. ACL, 2001.

[30] Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech & Language*, 14(4):283–332, 2000.

[31] Ciprian Chelba, Johan Schalkwyk, Thorsten Brants, Vida Ha, Boulos Harb, Will Neveitt, Carolina Parada, and Peng Xu. Query language modeling for voice search. In *Proc. SLT*, pages 127–132. IEEE, 2010.

[32] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google, 2013. URL http://arxiv.org/abs/1312.3005.

[33] Stanley Chen. Performance prediction for exponential language models. In *Proc. NAACL-HLT*, pages 450–458. ACL, 2009.

[34] Stanley Chen. Shrinking exponential language models. In *Proc. NAACL-HLT*, pages 468–476. ACL, 2009.

[35] Stanley Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[36] Stanley Chen, Lidia Mangu, Bhuvana Ramabhadran, Ruhi Sarikaya, and Abhinav Sethy. Scaling shrinkage-based language models. In *ASRU*, pages 299–304. IEEE Workshop, 2009.

[37] Xie Chen, Adam Eversole, Gang Li, Dong Yu, and Frank Seide. Pipelined back-propagation for context-dependent deep neural networks. In *Proc. ISCA INTERSPEECH*, 2012.

[38] Xie Chen, Mark Gales, Kate Knill, Catherine Breslin, Langzhou Chen, K.K. Chin, and Vincent Wan. An initial investigation of long-term adaptation for meeting transcription. In *Proc. Interspeech*, 2014.

[39] Xie Chen, Yongqiang Wang, Xunying Liu, Mark Gales, and P. C. Woodland. Efficient training of recurrent neural network language models using spliced sentence bunch. In *Proc. ISCA INTERSPEECH*, 2014.

[40] Xie Chen, Yongqiang Wang, Xunying Liu, Mark Gales, and Phil Woodland. Efficient GPU-based training of recurrent neural network language models using spliced sentence bunch. In *Proc. ISCA INTERSPEECH*, 2014.

[41] Xie Chen, Xunying Liu, Mark Gales, and Phil Woodland. CUED-RNNLM – an open-source toolkit for efficient training and evaluation of recurrent neural network language models. In *Proc. ICASSP*. IEEE, 2015.

[42] Xie Chen, Xunying Liu, Mark Gales, and Phil Woodland. Investigation of back-off based interpolation between recurrent neural network and n-gram language models. In *ASRU, IEEE Workshop on*, 2015.

[43] Xie Chen, Xunying Liu, Mark Gales, and Phil Woodland. Improving the training and evaluation efficiency of recurrent neural network language models. In *Proc. ICASSP*, 2015.

[44] Xie Chen, Xunying Liu, Mark Gales, and Phil Woodland. Recurrent neural network language model training with noise contrastive estimation for speech recognition. In *Proc. ICASSP*, 2015.

[45] Xie Chen, Tian Tan, Xunying Liu, Pierre Lanchantin, Moquan Wan, Gales Mark, and Phil Woodland. Recurrent neural network language model adaptation for multigenre broadcast speech recognition. In *Proc. ISCA INTERSPEECH*, 2015.

[46] Xie Chen, Xunying Liu, Yongqiang Wang, Mark Gales, and Phil Woodland. Efficient training and evaluation of recurrent neural network language models for automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2016.

[47] Xie Chen, Anton Ragni, Jake Vasilakes, Xunying Liu, and Mark Gales. Recurrent neural network language model for keywork search. In *Proc. ICASSP*, 2017.

[48] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[49] Philip Clarkson and Anthony Robinson. Language model adaptation using mixtures and an exponentially decaying cache. In *Proc. ICASSP*, pages 799–802. IEEE, 1997.

[50] Jia Cui, Xiaodong Cui, Bhuvana Ramabhadran, Janice Kim, Brian Kingsbury, Jonathan Mamou, Lidia Mangu, Michael Picheny, Tara Sainath, and Abhinav Sethy. Developing speech recognition systems for corpus indexing under the IARPA Babel program. In *Proc. ICASSP*, pages 6753–6757. IEEE, 2013.

[51] George Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[52] John N Darroch and Douglas Ratcliff. Generalized iterative scaling for log-linear models. *The annals of mathematical statistics*, 43(5):1470–1480, 1972.

[53] KH Davis, R Biddulph, and Stephen Balashek. Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6):637–642, 1952.

[54] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357–366, 1980.

[55] Wim De Mulder, Steven Bethard, and Marie-Francine Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, 2015.

[56] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Proc. NIPS*, pages 1223–1231, 2012.

[57] Renato DeMori and Marcello Federico. Language model adaptation. In *Computational models of speech pattern processing*, pages 280–303. Springer, 1999.

[58] Anoop Deoras, Tomáš Mikolov, and Kenneth Church. A fast re-scoring strategy to capture long-distance dependencies. In *Proc. EMNLP*, pages 1116–1127. ACL, 2011.

[59] Anoop Deoras, Tomas Mikolov, Stefan Kombrink, Martin Karafiát, and Sanjeev Khudanpur. Variational approximation of long-span language models for LVCSR. In *Proc. ICASSP*, pages 5532–5535. IEEE, 2011.

[60] Anoop Deoras, Tomas Mikolov, Stefan Kombrink, and Kenneth Church. Approximate inference: A sampling based modeling technique to capture complex dependencies in a language model. *Speech Communication*, 55(1):162–177, 2013.

[61] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *Proc. ACL*, 2014.

[62] Ahmad Emami and Lidia Mangu. Empirical study of neural network language models for Arabic speech recognition. In *ASRU, IEEE Workshop on*. IEEE, 2007.

[63] Gunnar Evermann and P. C. Woodland. Posterior probability decoding, confidence estimation and system combination. In *Proc. Speech Transcription Workshop*, volume 27. Baltimore, 2000.

[64] Gunnar Evermann, Ho Yin Chan, Mark Gales, Bin Jia, David Mrva, Phil Woodland, and Kai Yu. Training LVCSR systems on thousands of hours of data. In *Proc. of ICASSP*, volume 1, pages 209–212, 2005.

[65] Gunnar Evermann, Ho Yin Chan, Mark Gales, Bin Jia, David Mrva, Phil Woodland, and Kai Yu. Training LVCSR systems on thousands of hours of data. In *Proc. ICASSP*, pages 209–212. IEEE, 2005.

[66] Jonathan Fiscus, Jerome Ajot, Martial Michel, and John Garofolo. *The rich transcription 2006 spring meeting recognition evaluation*. Springer, 2006.

[67] Jonathan Fiscus, Jerome Ajot, and John Garofolo. The rich transcription 2007 meeting recognition evaluation. In *Multimodal Technologies for Perception of Humans*, pages 373–389. Springer, 2008.

[68] Mark Gales. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech & Language*, 12(2):75–98, 1998.

[69] Mark Gales. Semi-tied covariance matrices for hidden Markov models. *Speech and Audio Processing, IEEE Transactions on*, 7(3):272–281, 1999.

[70] Mark Gales. Cluster adaptive training of hidden Markov models. *Speech and Audio Processing, IEEE Transactions on*, 8(4):417–428, 2000.

[71] Mark Gales and Steve Young. The application of hidden Markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2008.

[72] Mark Gales, Kate Knill, Anton Ragni, and Shakti Rath. Speech recognition and keyword spotting for low resource languages: Babel project research at CUED. *Spoken Language Technologies for Under-Resourced Languages*, pages 16–23, 2014.

[73] John Garofolo, Christophe Laprun, Martial Michel, Vincent Stanford, and Elham Tabassi. The NIST meeting room pilot corpus. In *LREC*, 2004.

[74] J-L Gauvain and Chin-Hui Lee. Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains. *Speech and Audio Processing, IEEE Transactions on*, 2(2):291–298, 1994.

[75] Felix Gers, Jürgen Schmidhuber, et al. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[76] Daniel Gildea and Thomas Hofmann. Topic-based language models using em. In *Proc. of Eurospeech*, 1999.

[77] Irving Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

[78] Joshua Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

[79] Joshua Goodman. Classes for fast maximum entropy training. In *Proc. ICASSP*, volume 1, pages 561–564. IEEE, 2001.

[80] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proc. ICML*, pages 1764–1772, 2014.

[81] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[82] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proc. ICML*, pages 369–376. ACM, 2006.

[83] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *ASRU*, pages 273–278. IEEE Workshop, 2013.

[84] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

[85] Frantisek Grezl and Petr Fousek. Optimizing bottle-neck features for LVCSR. In *Proc. ICASSP*, pages 4729–4732. IEEE, 2008.

[86] František Grézl, Martin Karafiát, Stanislav Kontár, and J Cernocky. Probabilistic and bottle-neck features for LVCSR of meetings. In *Proc. ICASSP*, volume 4, pages IV–757. IEEE, 2007.

[87] Alexander Gutkin. Log-linear interpolation of language models. *Mémoire de DEA, University of Cambridge*, 2000.

[88] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361, 2012.

[89] Thomas Hain, Vincent Wan, Lukas Burget, Martin Karafiat, John Dines, Jithendra Vepa, Giulia Garau, and Mike Lincoln. The AMI system for the transcription of speech in meetings. In *Proc. ICASSP*, volume 4, pages IV–357. IEEE, 2007.

[90] Thomas Hain, Lukas Burget, John Dines, Philip N Garner, Frantisek Grezl, Asmaa El Hannani, Marijn Huijbregts, Martin Karafiat, Mike Lincoln, and Vincent Wan. Transcribing meetings with the AMIDA systems. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(2):486–498, 2012.

[91] Kenneth Heafield. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197. ACL, 2011.

[92] Kenneth Heafield, Ivan Pouzyrevsky, Jonathan Clark, and Philipp Koehn. Scalable modified kneser-ney language model estimation. In *Proc. ACL*, pages 690–696, 2013.

[93] Mike Henderson, Milica Gasic, Blaise Thomson, Pirros Tsiakoulis, Kai Yu, and Stephanie Young. Discriminative spoken language understanding using word confusion networks. In *Proc. SLT*, pages 176–181. IEEE, 2012.

[94] Hynek Hermansky. Perceptual linear predictive (plp) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[95] Hynek Hermansky, Daniel Ellis, and Sangita Sharma. Tandem connectionist feature extraction for conventional HMM systems. In *Proc. ICASSP*, volume 3, pages 1635–1638. IEEE, 2000.

[96] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[97] Geoffrey Hinton, Li Deng, Dong Yu, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[98] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[99] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[100] Takaaki Hori, Chiori Hori, Yasuhiro Minami, and Atsushi Nakamura. Efficient wfst-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(4):1352–1365, 2007.

[101] Bo-June Hsu. Generalized linear interpolation of language models. In *ASRU. IEEE Workshop*, pages 136–140, 2007.

[102] Xuedong Huang, James Baker, and Raj Reddy. A historical perspective of speech recognition. *Communications of the ACM*, 57(1):94–103, 2014.

[103] Zhiheng Huang, Geoffrey Zweig, Michael Levit, Benoit Dumoulin, Barlas Oguz, and Shawn Chang. Accelerating recurrent neural network training via two stage classes and parallelization. In *ASRU, IEEE Workshop on*. IEEE, 2013.

[104] Zhiheng Huang, Geoffrey Zweig, and Benoit Dumoulin. Cache based recurrent neural network language model inference for first pass speech recognition. In *Proc. ICASSP*, pages 6354–6358. IEEE, 2014.

[105] Kazuki Irie, Zoltán Tüske, Tamer Alkhouli, Ralf Schlüter, and Hermann Ney. LSTM, GRU, highway and a bit of attention: an empirical overview for language modeling in speech recognition. *Proc. ISCA Interspeech*, 2016.

[106] Rukmini Iyer and Mari Ostendorf. Modeling long distance dependence in language: Topic mixtures versus dynamic cache models. *Speech and Audio Processing, IEEE Transactions on*, 7(1):30–39, 1999.

[107] Shahab Jalalvand and Daniele Falavigna. Direct word graph rescoring using A* search and rnnlm. In *Proc. ISCA INTERSPEECH*, 2014.

[108] Adam Janin, Don Baron, Jane Edwards, Dan Ellis, David Gelbart, Nelson Morgan, Barbara Peskin, Thilo Pfau, Elizabeth Shriberg, Andreas Stolcke, et al. The ICSI meeting corpus. In *Proc. ICASSP*, volume 1, pages I–364. IEEE, 2003.

[109] Fred Jelinek. Speech recognition by statistical methods. *Proc. IEEE*, 64:532–556, 1976.

[110] Frederick Jelinek and Robert Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice*, 1980.

[111] Frederick Jelinek, Bernard Merialdo, Salim Roukos, and Martin Strauss. A dynamic language model for speech recognition. In *HLT*, volume 91, pages 293–295, 1991.

[112] Shihao Ji, SVN Vishwanathan, Nadathur Satish, Michael J Anderson, and Pradeep Dubey. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *arXiv preprint arXiv:1511.06909*, 2015.

[113] Penny Karanasou, Yongqiang Wang, Mark Gales, and Phil Woodland. Adaptation of deep neural network acoustic models using factorised i-vectors. In *Proc. ICSA INTERSPEECH*, pages 2180–2184, 2014.

[114] Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987.

[115] Brian Kingsbury. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *Proc. ICASSP*, pages 3761–3764. IEEE, 2009.

[116] Brian Kingsbury, Tara Sainath, and Hagen Soltau. Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization. In *Proc. ISCA INTERSPEECH*, 2012.

[117] Brian Kingsbury, Jia Cui, Xiaodong Cui, Mark Gales, Kate Knill, Jonathan Mamou, Lidia Mangu, David Nolden, Michael Picheny, Bhuvana Ramabhadran, et al. A high-performance Cantonese keyword search system. In *Proc. ICASSP*, pages 8277–8281. IEEE, 2013.

[118] Dietrich Klakow. Log-linear interpolation of language models. In *Proc. ICSLP*. IEEE, 1998.

[119] Reinhard Kneser and Hermann Ney. Improved clustering techniques for class-based statistical language modelling. In *Proc. EUROSPEECH*, page 1993.

[120] Reinhard Kneser and Hermann Ney. Improved backing-off for n-gram language modeling. In *Proc. ICASSP*. IEEE, 1995.

[121] Reinhard Kneser and Volker Steinbiss. On the dynamic adaptation of stochastic language models. In *Proc. ICASSP*, volume 2, pages 586–589. IEEE, 1993.

[122] Kate Knill, Mark Gales, Shakti Rath, Phil Woodland, Chao Zhang, and Shixiong Zhang. Investigation of multilingual deep neural networks for spoken term detection. In *ASRU*, pages 138–143. IEEE Workshop, 2013.

[123] Stefan Kombrink, Tomas Mikolov, Martin Karafiat, and Lukas Burget. Recurrent neural network based language modeling in meeting recognition. In *Proc. ISCA INTERSPEECH*, 2011.

[124] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1097–1105, 2012.

[125] Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12 (6):570–583, 1990.

[126] Roland Kuhn, J-C Junqua, Patrick Nguyen, and Nancy Niedzielski. Rapid speaker adaptation in eigenvoice space. *Speech and Audio Processing, IEEE Transactions on*, 8(6):695–707, 2000.

[127] Nagendra Kumar and Andreas G Andreou. *Investigation of silicon auditory models and generalization of linear discriminant analysis for improved speech recognition*. PhD thesis, Johns Hopkins University, 1997.

[128] Hong-Kwang Kuo, Ebru Arisoy, Ahmad Emami, and Paul Vozila. Large scale hierarchical neural network language models. In *Proc. ISCA INTERSPEECH*, 2012.

[129] Pierre Lanchantin, Peter J Bell, Mark Gales, Thomas Hain, Xunying Liu, Yanhua Long, Jennifer Quinnell, Steve Renals, Oscar Saz, Matthew Stephen Seigel, and Phil Woodland. Automatic transcription of multi-genre media archives. In *Proc. of the First Workshop on Speech, Language and Audio in Multimedia*, 2013.

[130] Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Trigger-based language models: A maximum entropy approach. In *Proc. ICASSP*, volume 2, pages 45–48. IEEE, 1993.

[131] Hai-Son Le, Ilya Oparin, Alexandre Allauzen, J Gauvain, and François Yvon. Structured output layer neural network language models for speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(1):197–206, 2013.

[132] Gwénolé Lecorvé and Petr Motlicek. Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition. Technical report, Idiap, 2012.

[133] Christopher J Leggetter and Phil Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech & Language*, 9(2):171–185, 1995.

[134] Hank Liao. Speaker adaptation of context dependent deep neural networks. 2013.

[135] Hank Liao and Mark Gales. Joint uncertainty decoding for noise robust speech recognition. In *Proc. ISCA INTERSPEECH*, pages 3129–3132, 2005.

[136] Xungying Liu, Xie Chen, Mark Gales, and Phil Woodland. Paraphrastic recurrent neural network language models. In *Proc. ICASSP*, pages 5406–5410. IEEE, 2015.

[137] Xunying Liu, Mark Gales, and C Woodland. Automatic complexity control for HLDA systems. In *Proc. ICASSP*, volume 1, pages I–132. IEEE, 2003.

[138] Xunying Liu, Mark Gales, and Phil Woodland. Context dependent language model adaptation. In *Proc. ICSA INTERSPEECH*, pages 837–840, 2008.

[139] Xunying Liu, Mark Gales, Jim L Hieronymus, and Phil Woodland. Language model combination and adaptation usingweighted finite state transducers. In *Proc. ICASSP*, pages 5390–5393. IEEE, 2010.

[140] Xunying Liu, Mark Gales, and Phil Woodland. Paraphrastic language models. In *Proc. ISCA INTERSPEECH*, 2012.

[141] Xunying Liu, Mark Gales, and Phil Woodland. Use of contexts in language model interpolation and adaptation. *Computer Speech & Language*, pages 301–321, 2013.

[142] Xunying Liu, Yongqiang Wang, Xie Chen, Mark Gales, and Phil Woodland. Efficient lattice rescoring using recurrent neural network language models. In *Proc. ICASSP*. IEEE, 2014.

[143] Xunying Liu, Federico Flego, Linlin Wang, Chao Zhang, Mark Gales, and Phil Woodland. The Cambridge University 2014 BOLT conversational telephone Mandarin Chinese LVCSR system for speech translation. In *Proc. ISCA INTERSPEECH*, volume 15, 2015.

[144] Xunying Liu, Xie Chen, Yongqiang Wang, Mark Gales, and Phil Woodland. Two efficient lattice rescoring methods using recurrent neural network language models. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(8):1438–1449, 2016.

[145] Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech & Language*, 14(4):373–400, 2000.

[146] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proc. ICML*, pages 1033–1040, 2011.

[147] Sven Martin, Jorg Liermann, and Hermann Ney. Algorithms for bigram and trigram word clustering1. *Speech Communication*, 24(1):19–37, 1998.

[148] Paul Mermelstein. Distance measures for speech recognition, psychological and instrumental. *Pattern Recognition and Artificial Intelligence*, 116:374–388, 1976.

[149] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Proc. ISCA INTERSPEECH*, pages 3771–3775, 2013.

[150] Tomas Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

[151] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. In *SLT*, 2012.

[152] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. In *SLT*, pages 234–239, 2012.

[153] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proc. ISCA INTERSPEECH*, 2010.

[154] Tomas Mikolov, Stefan Kombrink, Lukas Burget, J.H. Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Proc. ICASSP*. IEEE, 2011.

[155] Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukas Burget, and Jan Cernocky. Recurrent neural network language modeling toolkit. In *ASRU, IEEE Workshop*, 2011.

[156] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proc. NIPS*, pages 3111–3119, 2013.

[157] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proc. NAACL-HLT*, pages 746–751, 2013.

[158] Xavier Anguera Miro. *Robust speaker diarization for meetings*. PhD thesis, 2007.

[159] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *Proc. ICML*, 2012.

[160] Abdel-rahman Mohamed, Geoffrey Hinton, and Gerald Penn. Understanding how deep belief networks perform acoustic modelling. In *Proc. ICASSP*, pages 4273–4276. IEEE, 2012.

[161] Abdel-rahman Mohamed, Geoffrey Hinton, and Gerald Penn. Understanding how deep belief networks perform acoustic modelling. In *Proc. ICASSP*, pages 4273–4276. IEEE, 2012.

[162] Abdel-rahman Mohamed, Frank Seide, Dong Yu, Jasha Droppo, Andreas Stolcke, Geoffrey Zweig, and Gerald Penn. Deep bi-directional recurrent networks over spectral windows. ASRU, 2015.

[163] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311, 1997.

[164] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pages 246–252, 2005.

[165] David Mrva and Phil Woodland. A PLSA-based language model for conversational telephone speech. In *Proc. ISCA INTERSPEECH*, 2004.

[166] Joao Neto, Luís Almeida, Mike Hochberg, Ciro Martins, Luís Nunes, Steve Renals, and Tony Robinson. Speaker-adaptation for hybrid HMM-ANN continuous speech recognition system. 1995.

[167] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

[168] Hermann J Ney and Stefan Ortmanns. Dynamic programming search for continuous speech recognition. *Signal Processing Magazine, IEEE*, 16(5):64–83, 1999.

[169] Thomas Niesler, Edward Whittaker, and Phil Woodland. Comparison of part-of-speech and automatically derived category-based language models for speech recognition. In *Proc. ICASSP*, volume 1, pages 177–180. IEEE, 1998.

[170] JJ Odell, V Valtchev, Phil Woodland, and Steve Young. A one pass decoder design for large vocabulary recognition. In *Proc. HLT*, pages 405–410. ACL, 1994.

[171] Ilya Oparin, Martin Sundermeyer, Hermann Ney, and Jean-Luc Gauvain. Performance analysis of neural networks in combination with n-gram language models. In *Proc. ICASSP*, pages 5005–5008. IEEE, 2012.

[172] Junho Park, Xunying Liu, Mark Gales, and Phil Woodland. Improved neural network based language modelling and adaptation. In *Proc. ISCA INTERSPEECH*, 2010.

[173] Junho Park, Frank Diehl, Mark Gales, Marcus Tomalin, and Phil Woodland. The efficient incorporation of MLP features into automatic speech recognition systems. *Computer Speech & Language*, 25(3):519–534, 2011.

[174] Joris Pelemans, Noam Shazeer, and Ciprian Chelba. Pruning sparse non-negative matrix n-gram language models. In *Proc. ISCA INTERSPEECH*, pages 1433–1437, 2015.

[175] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of English words. In *Proc. ACL*, pages 183–190. ACL, 1993.

[176] Xuan-Hieu Phan and Cam-Tu Nguyen. Gibbslda++: A C/C++ implementation of latent Dirichlet allocation (LDA), 2007.

[177] Gerasimos Potamianos and Frederick Jelinek. A study of n-gram and decision tree letter language modeling methods. *Speech Communication*, 24(3):171–192, 1998.

[178] Daniel Povey. *Discriminative training for large vocabulary speech recognition*. PhD thesis, 2004.

[179] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The Kaldi speech recognition toolkit. In *ASRU, IEEE Workshop on*, 2011.

[180] Lawrence Rabiner and Biing-Hwang Juang. Fundamentals of speech recognition. 1993.

[181] Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286, 1989.

[182] Anton Ragni, Ed Dakin, Xie Chen, Mark Gales, and Kate Knill. Multi-language neural network language models. In *Proc. ISCA INTERSPEECH*, 2016.

[183] Steve Renals, Nelson Morgan, Hervé Bourlard, Michael Cohen, and Horacio Franco. Connectionist probability estimators in HMM speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 2(1):161–174, 1994.

[184] Steve Renals, Thomas Hain, and Herve Bourlard. Recognition and understanding of meetings the AMI and AMIDA projects. In *ASRU, IEEE Workshop on*, pages 238–247. IEEE, 2007.

[185] Korin Richmond, Robert Clark, and Susan Fitt. Robust LTS rules with the combilex speech technology lexicon. 2009.

[186] Tony Robinson and Frank Fallside. A recurrent error propagation network speech recognition system. *Computer Speech & Language*, 5(3):259–274, 1991.

[187] Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech & Language*, 10(3):187–228, 1996.

[188] Roni Rosenfield. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 2000.

[189] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[190] David Rumelhart, Geoffrey Hinton, and Ronald Williams. *Learning representations by back-propagating errors*. MIT Press, Cambridge, MA, USA, 1988.

[191] Hasim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proc. ICSA INTERSPEECH*, 2014.

[192] Hasim Sak, Andrew Senior, Kanishka Rao, Ozan Irsoy, Alex Graves, Françoise Beaufays, and Johan Schalkwyk. Learning acoustic frame labeling for speech recognition with recurrent neural networks. In *Proc. ICASSP*, pages 4280–4284. IEEE, 2015.

[193] George Saon, Hagen Soltau, David Nahamoo, and Michael Picheny. Speaker adaptation of neural network acoustic models using i-vectors. In *ASRU*, pages 55–59, 2013.

[194] George Saon, Steven Rennie, and Hong-Kwang J Kuo. The IBM 2016 English conversational telephone speech recognition system. *Proc. ISCA INTERSPEECH*, 2016.

[195] Holger Schwenk. Efficient training of large neural networks for language modelling. In *in Proceedings of IEEE International Joint Conference on Neural Networds*, pages 3059–3064. IEEE, 2002.

[196] Holger Schwenk. Continuous space language models. *Computer Speech & Language*, 21(3):492–518, 2007.

[197] Holger Schwenk. CSLM-a modular open-source continuous space language modeling toolkit. In *Proc. ICSA INTERSPEECH*, 2013.

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

[199] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Proc. ISCA INTERSPEECH*, pages 437–440, 2011.

[200] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. On parallelizability of stochastic gradient descent for speech DNNs. In *Proc. ICASSP*, 2014.

[201] Abhinav Sethy, Stanley Chen, Ebru Arisoy, and Bhuvana Ramabhadran. Unnormalized exponential and neural network language models. In *Proc. ICASSP*, pages 5416–5420. IEEE, 2015.

[202] Noam Shazeer, Joris Pelemans, and Ciprian Chelba. Skip-gram language modeling using sparse non-negative matrix probability estimation. *arXiv preprint arXiv:1412.1454*, 2014.

[203] Noam Shazeer, Joris Pelemans, and Ciprian Chelba. Sparse non-negative matrix language modeling for skip-grams. In *Proc. ISCA INTERSPEECH*, pages 1428–1432, 2015.

[204] Yangyang Shi. *Language Models With Meta-information*. PhD thesis, TU Delft, Delft University of Technology, 2014.

[205] Yangyang Shi, Mei-Yuh Hwang, Kaisheng Yao, and Martha Larson. Speed up of recurrent neural network language models with sentence independent subsampling stochastic gradient descent. In *Proc. ISCA INTERSPEECH*, 2013.

[206] Yangyang Shi, Matt Larson, and Catholijn M Jonker. K-component recurrent neural network language models using curriculum learning. In *ASRU*. IEEE, 2013.

[207] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu. Temporal kernel neural network language modeling. In *Proc. ICASSP*. IEEE, 2013.

[208] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu. Variance regularization of RNNLM for speech recognition. In *Proc. of ICASSP*, 2014.

[209] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu. Efficient one-pass decoding with NNLM for speech recognition. *Signal Processing Letterss on*, 21(4):377–381, 2014.

[210] Urmila Shrawankar and Vilas Thakare. Techniques for feature extraction in speech recognition system: A comparative study. *arXiv preprint arXiv:1305.1145*, 2013.

[211] Yujing Si, Ta Li, Jielin Pan, and Yonghong Yan. Prefix tree based n-best list re-scoring for recurrent neural netowrk language model used in speech recognition system. In *Proc. ISCA INTERSPEECH*, 2013.

[212] Andreas Stolcke, Barry Chen, Horacio Franco, Venkata Ramana Rao Gadde, Martin Graciarena, Mei-Yuh Hwang, Katrin Kirchhoff, Arindam Mandal, Nelson Morgan, Xin Lei, et al. Recent innovations in speech-to-text transcription at sri-icsi-uw. *Audio, Speech, and Language Processing, IEEE Transactions on*, 14(5):1729–1744, 2006.

[213] Andreas Stolcke et al. SRILM-an extensible language modeling toolkit. In *Proc. ISCA INTERSPEECH*, 2002.

[214] Hang Su, Gang Li, Dong Yu, and Frank Seide. Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription. In *Proc. ICASSP*, 2013.

[215] Yi Su, Frederick Jelinek, and Sanjeev Khudanpur. Large-scale random forest language models for speech recognition. In *Proc. ICSA INTERSPEECH*, pages 598–601, 2007.

[216] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Proc. ISCA INTERSPEECH*, 2012.

[217] Martin Sundermeyer, Ilya Oparin, Jean-Luc Gauvain, Ben Freiberg, Ralf Schluter, and Hermann Ney. Comparison of feedforward and recurrent neural network language models. In *Proc. ICASSP*, 2013.

[218] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. rwthlm — the RWTH Aachen university neural network language modeling toolkit. In *Proc. ICSA INTERSPEECH*, 2014.

[219] Martin Sundermeyer, Zoltán Tüske, Ralf Schlüter, and Hermann Ney. Lattice decoding and rescoring with long-span neural network language models. In *Proc. ISCA INTERSPEECH*, pages 661–665, 2014.

[220] Martin Sundermeyer, Hermann Ney, and Ralf Schluter. From feedforward to recurrent lstm neural networks for language modeling. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 23(3):517–529, 2015.

[221] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112, 2014.

[222] Pawel Swietojanski and Steve Renals. Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *Proc. SLT*, pages 171–176. IEEE, 2014.

[223] Yik-Cheung Tam and Tanja Schultz. Dynamic language model adaptation using variational bayes inference. In *Proc. ICSA INTERSPEECH*, pages 5–8, 2005.

[224] Yik-Cheung Tam and Tanja Schultz. Unsupervised language model adaptation using latent semantic marginals. In *Proc. ISCA INTERSPEECH*, 2006.

[225] Tian Tan, Yanmin Qian, Maofan Yin, Yimeng Zhuang, and Kai Yu. Cluster adaptive training for deep neural network. In *Proc. ICASSP*, pages 4325–4329. IEEE, 2015.

[226] Yee Whye Teh, Michael Jordan, Matthew Beal, and David M Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476), 2006.

[227] Ottokar TILK and Tanel ALUMAE. Multi-domain recurrent neural network language model for medical speech recognition. In *Human Language Technologies-The Baltic Perspective: Proceedings of the Sixth International Conference Baltic HLT 2014*, volume 268, page 149. IOS Press, 2014.

[228] Masaru Tomita. *Efficient parsing for natural language: a fast algorithm for practical systems*, volume 8. Springer Science & Business Media, 2013.

[229] Zoltan Tuske, David Nolden, Ralf Schluter, and Hermann Ney. Multilingual MRASTA features for low-resource keyword search and speech recognition systems. In *Proc. ICASSP*, pages 7854–7858. IEEE, 2014.

[230] Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. Decoding with large-scale neural language models improves translation. In *EMNLP*, pages 1387–1392, 2013.

[231] Dimitra Vergyri, Katrin Kirchhoff, Kevin Duh, and Andreas Stolcke. Morphology-based language modeling for Arabic speech recognition. In *Proc. ICSA INTERSPEECH*, volume 4, pages 2245–2248, 2004.

[232] Karel Veselỳ, Arnab Ghoshal, Lukás Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Proc. ICSA INTERSPEECH*, 2013.

[233] Haipeng Wang, Anton Ragni, Mark Gales, Kate Knill, Phil Woodland, and Chao Zhang. Joint decoding of tandem and hybrid systems for improved keyword spotting on low resource languages. In *Proc. ISCA INTERSPEECH*, 2015.

[234] Yongqiang Wang and Mark Gales. Tandem system adaptation using multiple linear feature transforms. 2013.

[235] Tsung-Hsien Wen, Aaron Heidel, Hung-Yi Lee, Yu Tsao, and Lin-Shan Lee. Recurrent neural network based personalized language modeling by social network crowdsourcing. In *Proc. ISCA INTERSPEECH*, 2011.

[236] Paul Werbos. Backpropagation through time: what it does and how to do it. *Proc. IEEE*, 78(10):1550–1560, 1990.

[237] Frank Wessel, Ralf Schluter, Klaus Macherey, and Hermann Ney. Confidence measures for large vocabulary continuous speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 9(3):288–298, 2001.

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

[239] Will Williams, Niranjani Prasad, David Mrva, Tom Ash, and Tony Robinson. Scaling recurrent neural network language models. *Proc. ICASSP*, 2015.

[240] Phil Woodland. Speaker adaptation for continuous density HMMs: A review. In *ISCA Tutorial and Research Workshop (ITRW) on Adaptation Methods for Speech Recognition*, 2001.

[241] Phil Woodland, Mark Gales, D Pye, and Steve Young. The development of the 1996 HTK broadcast news transcription system. In *DARPA speech recognition workshop*, pages 73–78. Morgan Kaufmann Pub, 1997.

[242] Phil Woodland, Xunying Liu, Yanmin Qian, Chao Zhang, Mark Gales, Penny Karanasou, Pierre Lanchantin, Linlin Wang, et al. Cambridge University transcription systems for the multi-genre broadcast challenge. 2015.

[243] Chunyang Wu and Mark Gales. Multi-basis adaptive neural network for rapid adaptation in speech recognition. In *Proc. ICASSP*, pages 4315–4319. IEEE, 2015.

[244] Youzheng Wu, Hitoshi Yamamoto, Xugang Lu, Shigeki Matsuda, Chiori Hori, and Hideki Kashioka. Factored recurrent neural network language model in ted lecture transcription. In *IWSLT*, pages 222–228, 2012.

[245] Peng Xu and Frederick Jelinek. Random forests in language modeling. In *Proc. EMNLP*, volume 4, pages 325–332, 2004.

[246] Hirojkmi Yamamoto and Yoshinori Sagisaka. Multi-class composite n-gram based on connection direction. In *Proc. ICASSP*, volume 1, pages 533–536. IEEE, 1999.

[247] Kaisheng Yao, Dong Yu, Frank Seide, Hang Su, Li Deng, and Yifan Gong. Adaptation of context-dependent deep neural networks for automatic speech recognition. In *Proc. SLT*, pages 366–369. IEEE, 2012.

[248] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. Recurrent neural networks for language understanding. In *Proc. ISCA INTERSPEECH*, pages 2524–2528, 2013.

[249] Takuya Yoshioka, Nobutaka Ito, Marc Delcroix, Atsunori Ogawa, Keisuke Kinoshita, Masakiyo Fujimoto, Chengzhu Yu, Wojciech J Fabian, Miquel Espi, Takuya Higuchi, et al. The ntt chime-3 system: Advances in speech enhancement and recognition for mobile multi-microphone devices. In *ASRU*, pages 436–443. IEEE, 2015.

[250] Steve Young, NH Russell, and JHS Thornton. *Token passing: a simple conceptual model for connected speech recognition systems*. Cambridge University Engineering Department Cambridge, UK, 1989.

[251] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, and et. al. The HTK book. *Cambridge University Engineering Department*, 2009.

[252] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Anton Ragni, Valtcho Valtchev, Phil Woodland, and Chao Zhang. The HTK book (for HTK version 3.5). *Cambridge University Engineering Department*, 2015.

[253] Dong Yu and Michael L Seltzer. Improved bottleneck features using pretrained deep neural networks. In *Proc. ISCA INTERSPEECH*, pages 237–240, 2011.

[254] Dong Yu, Kaisheng Yao, Hang Su, Gang Li, and Frank Seide. KL-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. Proc. ICASSP, 2013.

[255] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

[256] Pengyuan Zhang, Jian Shao, Jiang Han, Zhaojie Liu, and Yonghong Yan. Keyword spotting based on phoneme confusion matrix. In *Proc. of ISCSLP*, volume 2, pages 408–419, 2006.

[257] Geoffrey Zweig and Konstantin Makarychev. Speed regularization and optimality in word classing. In *Proc. ICASSP*, pages 8237–8241. IEEE, 2013.

Appendix A

Calculation of Gradient in Noise Contrastive Estimation

The aim of noise contrastive estimation training is to discriminate data generated from data distribution (i.e. RNNLM) from some known noise distribution. There is an prior assumption that the noise samples are k times more frequent than data samples.

Given a word w and its history h, let's denote the probability from data distribution (i.e. RNNLM) $P_{RNN}(w|h)$ and the probability from noise distribution $P_n(w|h)$. Hence, the posterior probability of w generated by RNNLM is,

$$P(w \in D|w,h) = \frac{P_{RNN}(w|h)}{P_{RNN}(w|h) + kP_n(w|h)} \tag{A.1}$$

The posterior probability of word w generated by noise distribution is,

$$P(w \in N|w,h) = 1 - P(w \in D|w,h) = \frac{kP_n(w|h)}{P_{RNN}(w|h) + kP_n(w|h)} \tag{A.2}$$

During NCE training, for each train sample w_i and its history h_i, k noise samples $\check{w}_{i,j}(j = 1,2,...k)$ are randomly sampled from a specified noise distribution (e.g. unigram distribution). The objective function is to minimize the negative log posterior probabilities over all samples, which can be written as,

$$J^{NCE}(\boldsymbol{\theta}) = -\frac{1}{N_w}\sum_{i=1}^{N_w}\left(\ln P(w_i \in D|w_i,h_i) + \sum_{j=1}^{k}\ln P(\check{w}_{i,j} \in N|\check{w}_{i,j},h_i)\right) \tag{A.3}$$

The gradient of the above objective function is,

$$\frac{\partial J^{NCE}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{1}{N_w}\sum_{i=1}^{N_w}\Big(\frac{\partial \ln P(w_i \in D|w_i,h_i)}{\partial \boldsymbol{\theta}} + \sum_{j=1}^{k}\frac{\partial \ln P(\breve{w}_{i,j} \in N|\breve{w}_{i,j},h_i)}{\partial \boldsymbol{\theta}}\Big) \tag{A.4}$$

Recalling Equation A.1 and A.2, the first term in the above equation is,

$$\begin{aligned}
\frac{\partial \ln P(w_i \in D|w_i,h_i)}{\partial \boldsymbol{\theta}} &= \frac{\partial \ln P(w_i \in D|w_i,h_i)}{\partial \boldsymbol{\theta}} \\
&= \frac{\partial \ln P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}} - \frac{\partial \ln(P_{RNN}(w_i|h_i)+kP_n(w_i|h_i))}{\partial \boldsymbol{\theta}} \\
&= \frac{1}{P_{RNN}(w_i|h_i)}\frac{\partial P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}} - \frac{1}{P_{RNN}(w_i|h_i)+kP_n(w_i|h_i)}\frac{\partial P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}} \\
&= \frac{kP_n(w_i|h_i)}{P_{RNN}(w_i|h_i)(P_{RNN}(w_i|h_i)+kP_n(w_i|h_i))}\frac{\partial P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}} \\
&= \frac{kP_n(w_i|h_i)}{(P_{RNN}(w_i|h_i)+kP_n(w_i|h_i))}\frac{\partial \ln P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}} \\
&= P(w_i \in N|w_i,h_i)\frac{\partial \ln P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}}
\end{aligned} \tag{A.5}$$

Similarly, the second term in Equation A.4 is,

$$\frac{\partial \ln P(\breve{w}_{i,j} \in N|\breve{w}_{i,j},h_i)}{\partial \boldsymbol{\theta}} = -P(\breve{w}_{i,j} \in D|\breve{w}_{i,j},h_i)\frac{\partial \ln P_{RNN}(\breve{w}_{i,j}|h_i)}{\partial \boldsymbol{\theta}} \tag{A.6}$$

Hence, the gradient of NCE objective function becomes,

$$\frac{\partial J^{NCE}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{1}{N_w}\sum_{i=1}^{N_w}\Big(P(w_i \in N|w_i,h_i)\frac{\partial \ln P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}} - \sum_{j=1}^{k}P(\breve{w}_{i,j} \in D|\breve{w}_{i,j},h_i)\frac{\partial \ln P_{RNN}(\breve{w}_{i,j}|h_i)}{\partial \boldsymbol{\theta}}\Big) \tag{A.7}$$

The gradient $\frac{\partial \ln P_{RNN}(w_i|h_i)}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \ln P_{RNN}(\breve{w}_{i,j}|h_i)}{\partial \boldsymbol{\theta}}$ can be computed via back-propagation easily.

Appendix B

Experiment on AMI IHM corpus

In this appendix, HTK and Kaldi toolkits were used to build acoustic models based on Hybrid system on the same data set to allow fair comparison The well-known AMI [27] corpus is chosen for experiment. Three microphone conditions were used to build the acoustic model and evaluate performance, which are IHM (Individual Headset Microphones), SDM (Single Distant Microphone), MDM (Multiple Distant Microphones) respectively. The Kaldi AMI recipe was adopted to build Kaldi system.[1]. The train, dev, and eval sets are consistent with the Kaldi AMI recipe. A 4-gram LM model was trained on text corpus including 12M words, which consists of 1M AMI transcriptions and 11M Fisher data(part 1). Sequence training were applied for the training of acoustic models for both of HTK and Kaldi Toolkits. The experimental results are shown in Table B.1. The WER results of HTK are reported on output from confusion network decoding and that of Kaldi are reported on MBR decoding.

According to the results in Tabel B.1, HTK gave better results with cross entropy training. For the MPE based sequence training, HTK and Kaldi gave comparable performances.

Table B.1 WER results of AMI eval set using HTK and Kaldi Toolkits. A 4-gram language model was used and CN (HTK), MBR (Kaldi) decoding were applied.

Train Crit	Toolkit	WER		
		IHM	MDM	SDM
CE	HTK	27.7	49.0	54.7
	Kaldi	28.3	50.5	57.2
MPE	HTK	26.3	47.0	52.4
	Kaldi	25.9	46.9	53.5

[1]The Kaldi recipe for AMI system can be found at
https://github.com/kaldi-asr/kaldi/tree/master/egs/ami/s5