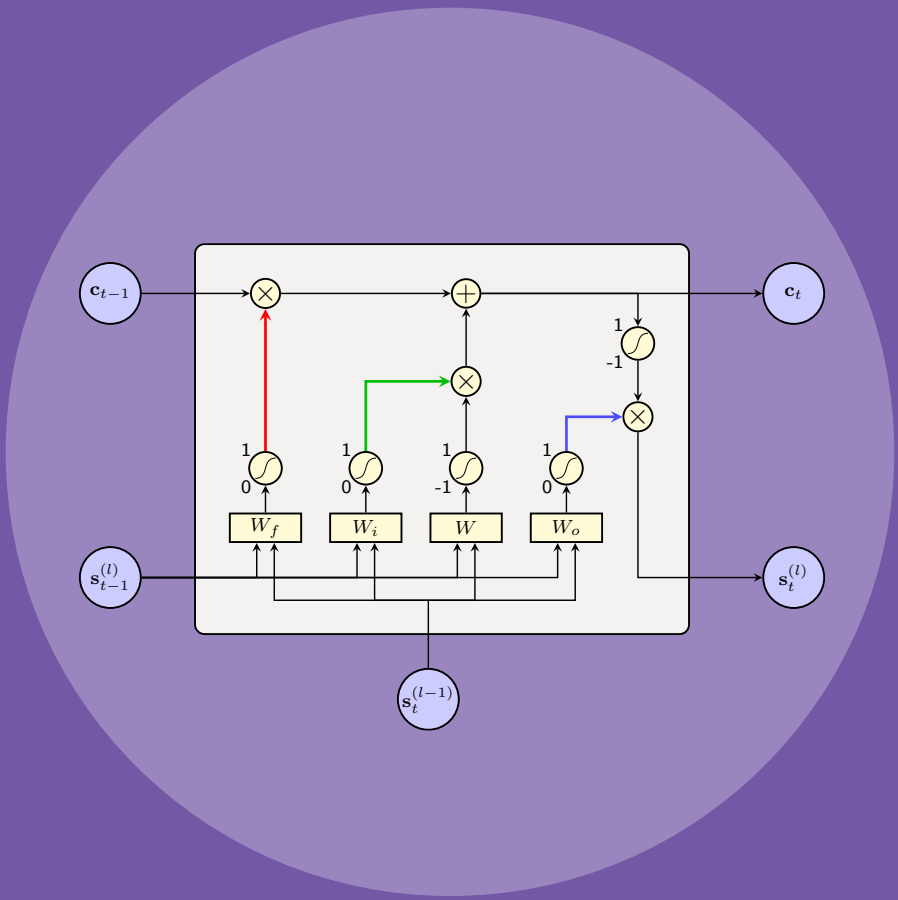
Department of Signal Processing and Acoustics

Modeling Conversational Finnish for Automatic Speech Recognition

Seppo Enarvi



Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 52/2018

Modeling Conversational Finnish for Automatic Speech Recognition

Seppo Enarvi

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Electrical Engineering, at a public examination held at the auditorium F239a of the HTH building on 3 May 2018 at 12.

Aalto University
School of Electrical Engineering
Department of Signal Processing and Acoustics
Speech Recognition Research Group

Supervising professor
Professor Mikko Kurimo

Thesis advisor
Dr. Sami Virpioja

Preliminary examiners
Professor Dietrich Klakow, Saarland University, Germany
Dr. Andreas Stolcke, Microsoft Research, USA

Opponent
Associate Professor Jan Černocký, Brno University of Technology, Czech Republic

Aalto University publication series
DOCTORAL DISSERTATIONS 52/2018

ISBN 978-952-60-7907-3 (printed)
ISBN 978-952-60-7908-0 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-7908-0

Unigrafia Oy
Helsinki 2018

Finland

441 697
Printed matter

Author
Seppo Enarvi

Name of the doctoral dissertation
Modeling Conversational Finnish for Automatic Speech Recognition

Publisher School of Electrical Engineering

Unit Department of Signal Processing and Acoustics

Series Aalto University publication series DOCTORAL DISSERTATIONS 52/2018

Field of research Speech and Language Technology

Manuscript submitted 19 October 2017 **Date of the defence** 3 May 2018

Permission to publish granted (date) 5 January 2018 **Language** English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

The accuracy of automatic speech recognizers has been constantly improving for decades. Aalto University has developed automatic recognition of Finnish speech and achieved very low error rates on clearly spoken standard Finnish, such as news broadcasts. Recognition of natural conversations is much more challenging. The language that is spoken in Finnish conversations also differs in many ways from standard Finnish, and its recognition requires data that has previously been unavailable.

This thesis develops automatic speech recognition for conversational Finnish, starting by collection of training and evaluation data. For language modeling, large amounts of text are collected from the Internet, and filtered to match the colloquial speaking style. An evaluation set is published and used to benchmark the progress in conversational Finnish speech recognition. The thesis addresses many difficulties that arise from the fact that the vocabulary that is used in Finnish conversations is very large. Using deep neural networks for acoustic modeling and recurrent neural networks for language modeling, accuracy that is already useful in practical applications is achieved in conversational speech recognition.

Keywords automatic speech recognition, language modeling, word classes, artificial neural networks, data collection

ISBN (printed) 978-952-60-7907-3 **ISBN (pdf)** 978-952-60-7908-0

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2018

Pages 185 **urn** http://urn.fi/URN:ISBN:978-952-60-7908-0

Tekijä
Seppo Enarvi

Väitöskirjan nimi
Suomen puhekielen mallintaminen automaattista puheentunnistusta varten

Julkaisija Sähkötekniikan korkeakoulu

Yksikkö Signaalinkäsittelyn ja akustiikan laitos

Sarja Aalto University publication series DOCTORAL DISSERTATIONS 52/2018

Tutkimusala Puhe- ja kieliteknologia

Käsikirjoituksen pvm 19.10.2017 **Väitöspäivä** 03.05.2018

Julkaisuluvan myöntämispäivä 05.01.2018 **Kieli** Englanti

☐ **Monografia** ☒ **Artikkeliväitöskirja** ☐ **Esseeväitöskirja**

Tiivistelmä

Automaattisen puheentunnistuksen tarkkuus on jatkuvasti parantunut viimeisten vuosikymmenien aikana. Aalto-yliopistossa on kehitetty automaattista puheentunnistusta suomen kielelle ja päästy hyvin pieniin virheprosentteihin selkeästi puhutun kirjakielen tunnistuksessa, esimerkiksi uutislähetyksistä. Luonnollisten keskustelujen tunnistaminen on paljon haastavampaa. Suomen puhekieli eroaa myös monella tavalla kirjakielestä, ja sen tunnistamiseen tarvitaan tietoaineistoa, jota ei aikaisemmin ole ollut saatavilla.

Tämä väitöskirja kehittää automaattista puheentunnistusta suomen puhekielelle, alkaen opetus- ja testiaineiston keräämisestä. Kielen mallintamista varten Internetistä kerätään suuri määrä tekstiä ja aineisto suodatetaan vastaamaan puhekielen tyyliä. Testiaineisto julkaistaan ja sitä käytetään kriteerinä, kun arvioidaan suomen kielen keskustelumuotoisen puheen tunnistuksen kehitystä. Väitöskirjassa tutkitaan monia ongelmia jotka juontuvat siitä, että sanasto jota käytetään suomenkielisissä keskusteluissa on todella iso. Kun syviä neuroverkkoja käytetään akustiseen mallinnukseen ja takaisinkytkettyjä neuroverkkoja käytetään kielen mallinnukseen, saavutetaan keskustelupuheen tunnistuksessa tarkkuus joka on jo kelvollinen käytännön sovelluksiin.

Avainsanat automaattinen puheentunnistus, kielen mallintaminen, sanaluokat, neuroverkot, tiedonkeruu

ISBN (painettu) 978-952-60-7907-3 **ISBN (pdf)** 978-952-60-7908-0

ISSN-L 1799-4934 **ISSN (painettu)** 1799-4934 **ISSN (pdf)** 1799-4942

Julkaisupaikka Helsinki **Painopaikka** Helsinki **Vuosi** 2018

Sivumäärä 185 **urn** http://urn.fi/URN:ISBN:978-952-60-7908-0

Preface

I started working on this thesis in 2012, in the Speech Recognition Research Group of the Department of Information and Computer Science at Aalto University School of Science. The group moved in 2013 to the Department of Signal Processing and Acoustics, which is part of the School of Electrical Engineering. I am thankful for the opportunity to work at Aalto University, and of the funding I received from external sources. I received scholarships from Finnish Cultural Foundation (Suomen Kulttuurirahasto) and Kone Foundation (Koneen Säätiö). This research was also partially funded through grants awarded by the Academy of Finland (Suomen Akatemia). The research involved many compute-intensive experiments that relied on the computational resources provided by the Aalto Science-IT project.

I would like to thank my colleagues who I have had the pleasure to work with in the speech group. I am in especially great gratitude to my supervisor Prof. Mikko Kurimo, who has guided my work, aided in writing the articles, and was extremely helpful in arranging the funding. I appreciate all the support I got from Janne Pylkkönen while I was starting speech recognition research. I am thankful to Sami Virpioja for examining the thesis and our co-operation in writing a journal article. I had interesting discussions, collaborated in writing articles, and received kindly feedback for my thesis from Peter Smit and Matti Varjokallio. I am grateful also to André Mansikkaniemi for our collaboration in writing a journal article.

In addition to the members of the speech group at Aalto University, I had the pleasure to collaborate with Tanel Alumäe and Ottokar Tilk from Tallinn University of Technology in writing a journal article. I had a very inspiring visit to the International Computer Science Institute in 2012, which was funded by Helsinki Institute for Information Technology. I am

thankful for getting to know such a diverse group of people who taught me a lot of things about research, speech recognition, and playing table football. Afterwards I had valuable discussions with Oriol Vinyals regarding my next conference paper.

My family and friends always seemed to know I am going to pursue a Ph.D. even before I knew it myself. Now I realize how important their encouragement has been. During the long and intense time that I was working on this thesis I also understood how lucky I was to have the support that I have had from my life partner.

Helsinki, February 24, 2018,

Seppo Enarvi

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Seppo Enarvi and Mikko Kurimo. A Novel Discriminative Method for Pruning Pronunciation Dictionary Entries. In *Proceedings of the 7th International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, Cluj-Napoca, Romania, pages 113–116, October 2013.

II Seppo Enarvi and Mikko Kurimo. Studies on Training Text Selection for Conversational Finnish Language Modeling. In *Proceedings of the 10th International Workshop on Spoken Language Translation (IWSLT)*, Heidelberg, Germany, pages 256–263, December 2013.

III Mikko Kurimo, Seppo Enarvi, Ottokar Tilk, Matti Varjokallio, André Mansikkaniemi, and Tanel Alumäe. Modeling under-resourced languages for speech recognition. *Language Resources and Evaluation*, volume 51, issue 4, pages 961–987, December 2017.

IV Seppo Enarvi and Mikko Kurimo. TheanoLM — An Extensible Toolkit for Neural Network Language Modeling. In *Proceedings of the 17th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, San Francisco, CA, USA, pages 3052–3056, September 2016.

V Seppo Enarvi, Peter Smit, Sami Virpioja, and Mikko Kurimo. Automatic Speech Recognition with Very Large Conversational Finnish and

Estonian Vocabularies. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, volume 25, issue 11, pages 2085–2097, November 2017.

Author's Contribution

Publication I: "A Novel Discriminative Method for Pruning Pronunciation Dictionary Entries"

The author invented and implemented the algorithm described in the article, and conducted the experiments. The author was also the main writer of the article.

Publication II: "Studies on Training Text Selection for Conversational Finnish Language Modeling"

The author designed and implemented the data collection and filtering methods used in the article. The author was also the main contributor in analyzing the results and writing the article.

Publication III: "Modeling under-resourced languages for speech recognition"

The author contributed the experiments on selecting conversational text from the Internet. The author implemented the data selection algorithms, and improved them to work efficiently with large data sets and morphologically rich languages. The author also provided the lattice-based method for pruning pronunciations of foreign proper names. The author wrote mainly the sections related to data selection.

Publication IV: "TheanoLM — An Extensible Toolkit for Neural Network Language Modeling"

The author implemented the language modeling toolkit and performed the experiments. The author was also the main contributor in analyzing the results and writing the article.

Publication V: "Automatic Speech Recognition with Very Large Conversational Finnish and Estonian Vocabularies"

The author contributed the experiments on clustering words into classes and designed the rule-based method for clustering Finnish words. The author implemented the neural network language modeling methods and lattice decoding, and performed the lattice rescoring experiments. The author was also the main contributor in analyzing the results and writing the article.

List of Abbreviations

ASR	automatic speech recognition
BPTT	backpropagation through time
CBOW	continuous bag-of-words
CEC	constant error carousel
CER	character error rate
CSG	continuous skip-gram
CTC	connectionist temporal classification
DNN	deep neural network
EM	expectation maximization
FST	finite-state transducer
GMM	Gaussian mixture model
GPU	graphics processing unit
HMM	hidden Markov model
HTML	hypertext markup language
IPA	International Phonetic Alphabet
LER	letter error rate
LSTM	long short-term memory
MAP	maximum a posteriori
MDL	minimum description length
MFCC	mel-frequency cepstral coefficient
ML	maximum likelihood
MMI	maximum mutual information
MSE	mean squared error
NCE	noise-contrastive estimation
NIST	National Institute of Standards and Technology
NNLM	neural network language model
OOS	out of shortlist
OOV	out of vocabulary

RNN	recurrent neural network
SGD	stochastic gradient descent
TDNN	time delay neural network
tf–idf	term frequency–inverse document frequency
WER	word error rate
XML	extensible markup language

List of Symbols and Notations

$\mathbf{a}^{(l)}$	a vector of preactivations of layer l in a neural network
$\mathbf{b}^{(l)}$	the bias vector of layer l in a neural network
$c(\mathbf{w})$	the number of occurrences of n-gram w in training data
$\mathcal{C}$	a cost function for an optimization task
$D_{\mathrm{KL}}(p \parallel p')$	relative entropy (Kullback–Leibler divergence) from probability distribution p to p'
$\mathrm{E}[X]$	expectation of the random variable X
$\mathbf{h}^n$	the history (context) of length n in a sequence of words
$\mathcal{H}(p)$	entropy of the probability distribution p
$\mathcal{H}(Y \mid X)$	conditional entropy of the random variable Y given the random variable X
$\mathcal{H}(p, p')$	cross entropy between two probability distributions p and p'
$\mathcal{H}(\mathbf{x}, p)$	empirical cross entropy between the distribution of the data x and the probability distribution p
$\mathcal{L}(\theta)$	likelihood of the model parameters θ
N	number of words
$p(x)$	a probability function
$PP(p)$	perplexity of the probability distribution p
$PP(\mathbf{x}, p)$	empirical perplexity of the probability distribution p on the data x
$\mathbf{s}^{(l)}$	the hidden state of layer l in a neural network
$\mathbf{w}$	a sequence of words / an n-gram
$W^{(l)}$	the weight matrix of layer l in a neural network
z	a latent variable in an HMM or a mixture model
α	a hyperparameter controlling the objective of the Morfessor algorithm
$\beta(\mathbf{h}^{n-1})$	a back-off weight of an n-gram language model

δ_{ij}	the Kronecker delta function
$\boldsymbol{\Delta}^{(l)}$	error vector in layer l during backpropagation
$\zeta(x)$	the softplus function
η	learning rate in numerical optimization
θ	model parameters
λ	an interpolation weight / a Lagrange multiplier in constrained optimization
$\sigma(x)$	the logistic sigmoid function
$\phi(\mathbf{a})$	an activation function in a neural network
$\mathbf{x} \odot \mathbf{y}$	Hadamard (elementwise) product of vectors $\mathbf{x}$ and $\mathbf{y}$

1. Introduction

1.1 Transcribing Finnish Conversations

Speech recognition is the task of writing a text transcript of what was said in an audio signal. During the past five decades, speech recognition has developed from classifying individual words to transcribing continuous speech. Initially, the vocabulary that the systems were able to recognize consisted of just 10 words, but modern systems are able to recognize hundreds of thousands to millions of different words, or even text that is not limited to a certain set of words by using subword or letter models. The early systems were speaker dependent, meaning that they worked only for the same speaker that was used to train the recognizer, but modern speaker independent systems can generalize to the speech of any speaker.

Automatic speech recognizers have already since the turn of the century worked well for planned English, such as broadcast news. Another task where automatic speech recognition has shined is dictation of e.g. medical reports. When the topic of the recognized speech is limited to a very specific domain, and the statistical models used by the speech recognizer can be adapted to the speaking style of the specific speaker, accuracy of automatic speech recognition can approach that of a human transcriptionist. In these applications the speaker also tries to speak as clearly as possible.

On the other hand, recognition of spontaneous conversations has remained a challenge. Also, the research has clearly concentrated on English language, and other languages usually have far less resources. For example, in the Rich Transcription Evaluation implemented by NIST in 2003 [71], a 9.9 % word error rate (WER) was obtained in transcription of English language broadcast news. 23.8 % WER was obtained for the Switchboard database, which consists of recordings of telephone discus-

sions of proposed topics. Speech recognition performance on Chinese and Arabic data was significantly worse.

A huge leap forward in conversational speech recognition accuracy happened around 2010 by the introduction of deep neural network (DNN) acoustic models. With a 7 layers deep feedforward network, a reduction in WER from 27.4 % to 18.5 % was reported on the Switchboard database [80]. A similar improvement was not observed in a broadcast news task [35].

Speech recognition accuracy on read Finnish was already good before I started to work on this thesis. In 2006, 7.0 % WER was achieved on audio books, but due to the lack of data, a speaker dependent model was used [52]. In 2009, speaker independent models were trained, one on clean speech from the Speecon corpus and one on the SpeechDat telephone speech corpus [75]. Word error rates of 13.7 % and 22.3 % were obtained on these tasks respectively, using maximum likelihood training, and even better results with discriminative training. The success is to some extent attributed to the use of subword language models created using the Morfessor method [37]. The results are very good, considering that the WER numbers are generally higher because the vocabulary is unlimited.

On the other hand, no research was done on recognizing conversational Finnish. There are many tasks where this would be useful, for example automatic transcription of meetings, and subtitling of broadcast conversations. There are various reasons why transcribing conversations is more difficult than recognizing planned speech. The pronunciation of words, as well as grammar used in conversations can be different and less coherent. Conversational speech is not organized in sentences in the same way as formal speech is. Instead, speech often flows continuously with filler words used to signal a pause. The rate of speech can vary and disfluencies can make recognition difficult. With all these differences, it would be important to have training data that consists of actual spontaneous conversations.

When the work on this thesis began in 2012, only a few small corpora of conversational Finnish were available. In Publication II the first serious attempts of recognizing conversational Finnish speech were made. Collection of a conversational Finnish corpus was started at Aalto University, and part of the corpus was dedicated as development and evaluation data. Written conversations were collected from the Internet for language modeling. By combining all the data at hand, 55.6 % WER was obtained.

The collected data sets are not constrained to any particular topics.

Colloquial Finnish also differs substantially from the standard language in vocabulary, and many words have alternative forms in which they can be written and pronounced. In Publication II we showed that this amounts to the vocabulary used in conversational Finnish text being larger than the vocabulary size in the same amount of standard Finnish text.

Two approaches to modeling different pronunciations in Finnish language are discussed in this thesis. Traditionally alternative pronunciations for words have been specified in a pronunciation dictionary. On the other hand, language models can be trained on the conversational Finnish data, where different pronunciations are written out as different word forms. Essentially this means that the model estimates probabilities for a sequence of pronunciations instead of a sequence of words. In practice the approach is problematic due to the vocabularies being bigger and the data even more sparse than standard Finnish data. One has to also consider how to compute the recognition accuracy.

1.2 Scope and Contributions of the Thesis

This thesis is founded on research that has advanced automatic speech recognition since the 1950s. Most importantly, Finnish language speech recognition research has been carried out at Helsinki University of Technology, which was merged into Aalto University in 2010. As a result of the earlier research, a Finnish language speech recognizer has been developed that works well on clear speech. Speech utterances in the training and evaluation data have been planned or the speakers have responded to given situations, and the language has been close to standard Finnish.

The language and conditions in natural conversations are often more versatile, making the speech difficult to recognize. In this thesis I have not attempted to further improve the recognition of planned speech, but concentrated on spontaneous conversations on unconstrained topics. The data set I have used is especially problematic, because the speakers use colloquial Finnish, which differs quite significantly from standard Finnish. A simple reason why colloquial Finnish could not be recognized well, is that there were no large corpora specifically targeted to colloquial Finnish. An important contribution of this work is the collection of suitable training data for the statistical models necessary for automatic speech recognition.

DSPCON corpus has been collected during 2013–2016 and contains conversations between students of the basic course in digital signal processing

at Aalto University. Part of the corpus has been dedicated for evaluation, and for computing the error rate, alternative word forms have been added to the transcripts. The same evaluation data has been used throughout the thesis, so that the progress in conversational Finnish speech recognition can be followed.

While DSPCON contains valuable acoustic data for modeling pronunciation of Finnish in a conversational setting, the number of sentences is small compared to the amount of text required for properly modeling a language without limiting to any particular topic or context. Huge amounts of text can be found from the Internet, but since we are interested in modeling conversational Finnish, the text should match the conversational speaking style as closely as possible. Methods for selecting matching text based on a small sample of transcribed Finnish conversations are developed in this thesis (Publications II and III). While such algorithms have existed before, the contribution of this thesis is specifically in making these algorithms work with Finnish data and perform efficiently with large amounts of data.

The rest of the thesis is devoted to modeling efforts. The thesis proposes a discriminative pruning method for optimizing a pronunciation dictionary (Publication I). The method is especially useful when a large number of pronunciations are generated automatically. It is first tested on a hand-crafted pronunciation dictionary and later applied to adapting models with automatically generated pronunciations for foreign names (Publication III).

Colloquial Finnish contains a lot of variation in pronunciation. A consequence of the phonemic orthography (writing system) of Finnish language is that a writer may alter the spelling of a word according to how its pronunciation would change. Usually the colloquial writing is used in informal written conversations, such as e-mails and conversation sites on the Internet. Conversational Finnish models in this thesis are trained on text that has been downloaded from conversation sites, and often is written in an informal style that mimics the colloquial pronunciation. As a consequence, the vocabulary is very large (Publication II).

This thesis attempts to solve the problems that follow from the very large vocabulary and data sparsity (Publication V). Different methods for clustering words into classes are evaluated. A rule-based method is proposed for clustering word forms that correspond to different pronunciations of the same word. Another interesting approach is to segment words into smaller units. Subword units created using Morfessor are evaluated, and found

to outperform word models when recurrent neural networks are used for language modeling.

Neural networks are used for language modeling in the last two articles (Publication IV and V). Different approximations for neural networks are evaluated that make it possible to use large vocabularies in neural network language models. A novel method for weighting data sources in training is tested. All the developed methods are published in TheanoLM toolkit, including different optimization methods, layer types, sampling-based objectives, and support for n-best list and word lattice rescoring.

Finally, we attempt to build as good speech recognition system as possible, using deep neural network acoustic models and complex recurrent neural network language models (Publication V). As a result, a word error rate of 27.1 % is achieved in conversational Finnish speech recognition using subword units. The experiments are repeated on a conversational Estonian speech recognition task, and again a state-of-the-art result, 21.9 % word error rate is reached.

1.3 Structure of the Thesis

Chapter 2 gives an overview of the speech recognition problem, elaborating on some issues that are relevant in particular for recognizing conversational Finnish. It first introduces different approaches that have been taken for transcribing speech, and then presents in more detail the most popular, hidden Markov model (HMM) based, speech recognition framework, which is used in this thesis. Then it explains the problems that are encountered when recognizing conversational Finnish speech due to pronunciation variation in conversations and agglutination in Finnish language.

The main focus in this thesis is on language modeling. Chapter 3 describes the well-established techniques for training n-gram language models in detail. Then relevant variations of the standard model are presented: class-based language models, maximum entropy models, subword models, and variable-order training.

Chapter 4 starts by explaining the basics of modeling with artificial neural networks, with focus on language modeling. The problems encountered in deep networks and in models with a large number of outputs are discussed. The chapter aims to describe the techniques that were used in the neural network models in the publications with more detail than what

was possible in the articles. Some details on implementing TheanoLM are also given.

Chapter 5 describes the work that was done on recording speech data and collecting text from the Internet. The method that was used for crawling web sites is explained. The steps taken to normalize Finnish text are listed, and an overview of the text filtering methods is given.

Chapter 6 concludes the results of this thesis and suggests some directions for future research.

2. Automatic Speech Recognition for Conversational Finnish

2.1 Approaches to Speech Recognition

Automatic recognition of speech is a rather complex machine learning task. The aim is to translate a speech signal to text. This is a classification task, but the search space is very large, as the length of the word sequence is not limited, and the vocabulary can be large. Another thing that makes this task more difficult than a typical classification task is that the alignment of the text and the acoustic signal is unknown.

Early speech recognition systems in the attempted to recognize isolated words. A digit recognizer [20] and a vowel recognizer [26] used the output of filter banks to classify an utterance to one of the 10 words in the vocabulary. An approach that was studied in the 1960s and 1970s was to match a speech sample to reference templates. A template of each vocabulary word was required. Accuracy was improved by dividing the signal into short fixed-length segments and aligning the features extracted from the segments to those of the template using dynamic time warping. A vocabulary of 203 words was used already in 1970 [93].

Dynamic time warping finds a mapping between the time scale of the input signal and that of the reference template (see Figure 2.1). This fixed mapping does not account for the uncertainties that arise from the variability in speaking style and recording conditions. This approach also does not scale for continuous speech. At the same time, the use of hidden Markov models to model speech signal emerged [42]. However, it was not until the 1980s when the theory was widely spread among researchers and fully developed to combine Gaussian mixture emission probabilities.

The HMM approach [53] also divides the signal into fixed-length (e.g. 25 ms) segments called frames. It is assumed that the vocal tract generating

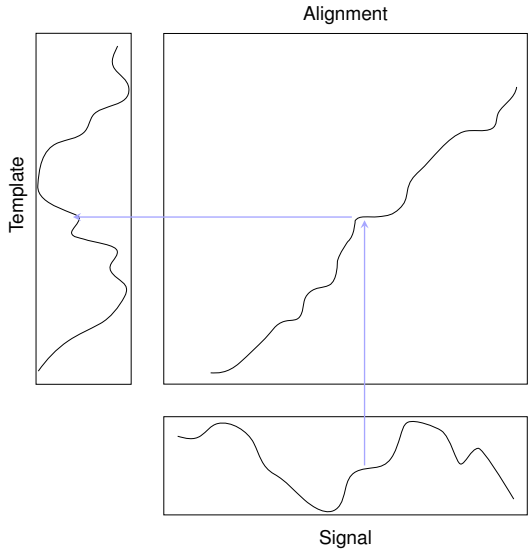


Figure 2.1. Dynamic time warping aligns the signal with the reference templates, in order to find the best matching template.

the speech is at any given time at a certain state. The state sequence $\{z_t\}$ is a sequence of latent variables, meaning that they cannot be observed. The acoustic features $\{\mathbf{o}_t\}$ are observed variables, extracted from each audio frame. The acoustic observations are assumed to be generated with certain *emission probability* distribution that depends on the current state. This is depicted in Figure 2.2. A *transition probability* is associated between any two states. The state may change according to the transition probabilities, or stay the same in the next frame.

HMMs solve the alignment problem neatly, because they do not assume anything about the length of the audio signal. Individual HMMs can be concatenated, which makes continuous speech recognition possible. The signal is no longer segmented into isolated words, rather the speech recognizer is able to evaluate different hypotheses with different alignments, which accounts for uncertainties in the model. Larger vocabularies can be used by constructing words from HMMs of subword units, such as phonemes. A pronunciation dictionary maps words to a sequence of subword HMM models. The probabilistic formulation enables the combination of multiple sources of information. A separate probabilistic model of the language is used to predict which words commonly follow each other. These advances led to the adoption of HMMs by virtually every speech recognition system at the time.

The idea of using neural networks for modeling speech surfaced already

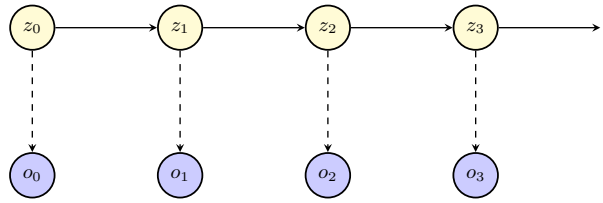


Figure 2.2. A hidden Markov model assumes that the vocal tract is at any given time in state z_t. The state cannot be observed, but the observed acoustic features, o_t, are generated with certain emission probability distribution that depends on the current state.

at the end of the 1980s. In a hybrid approach, a neural network is used to estimate the emission probabilities, while the HMM framework is still used for decoding continuous speech as a sequence of words [10]. In the beginning simple feedforward networks were used. It took another two decades before the technology and training algorithms were advanced enough to use complex neural networks with many hidden layers that would clearly outperform traditional Gaussian mixture emission probabilities even with large amounts of data [35].

Although the HMM framework is still most widely used, it is not the most elegant. The pipeline consists of many components that are hand-engineered for a specific purpose. The framework makes assumptions about the model structure that may not be optimal. For example, a well known weakness of HMMs is that the output observations depend only on the current state, not on the previous observation or the previous states. Recent advances in neural networks have proven that neural networks are very good at learning complex representations from data automatically, spurring development of *end-to-end* speech recognition [32]. The idea is to train a single recurrent neural network that predicts words given the acoustic observations.

An end-to-end speech recognizer can operate on the raw audio signal, although usually feature extraction is performed first to compress the input. In a typical classification setting, a recurrent neural network (RNN) outputs one class for each input. In a speech recognizer, an RNN could output a sequence of letters, whose length is the number of frames. However, it is not clear how to align the letters to form words. For example, we need to decide whether a letter repeating in consecutive frames corresponds to one or two letters in the word.

One method for adapting RNNs to unaligned input and output sequences is called connectionist temporal classification (CTC) [31]. A blank label is

included in the output alphabet. Any alignment can be represented as a sequence that is as long as the input sequence, using the blank symbol to represent a letter boundary. Given an alignment, the probability of the utterance can be computed as the product of the frame probabilities. When the RNN is trained, the probability of an utterance is defined as the sum of the probabilities over the possible alignments. This training objective can be realized using a dynamic programming algorithm.

In end-to-end speech recognition, the RNN uses history in addition to the current input for predicting the output at any time. In principle, given enough training data, it could learn dependencies between words. However, in practice a language model is still needed for good results. When enough training data is available, an end-to-end speech recognizer may achieve as good or even better performance than state-of-the-art HMM-based systems.

2.2 Speech Recognition Using the HMM Framework

The probabilistic HMM framework allows combination of information from multiple sources. Typically these include an acoustic model, a language model, and a pronunciation dictionary. The pronunciation dictionary maps words to one or more different pronunciations, possibly with different prior probabilities. A pronunciation is defined as a sequence of HMMs, each of which is defined in the acoustic model. The language model scores word sequences based on which words frequently occur together in the target language.

A choice has to be made on what the basic unit of speech modeled by each HMM of the acoustic model is. Having a separate HMM for each word is not realistic for large vocabularies, and would require a new model to be trained every time a word is added to the vocabulary. The smallest units of speech are called *phones*. A closely related concept is *phoneme*, a group of phones that are semantically equivalent in a language. Phonemes are a good candidate for the basic acoustic modeling unit. However, their pronunciation often changes based on the neighboring phonemes, which is why most current speech recognition systems, including AaltoASR and Kaldi, use *context dependent* phoneme models.

The speech recognition pipeline is illustrated in Figure 2.3. In the first step, features are extracted from the speech signal. A good set of features is as compact as possible, while still being able to discriminate between the speech sounds. Most speech recognizers, as well as a broad range of other

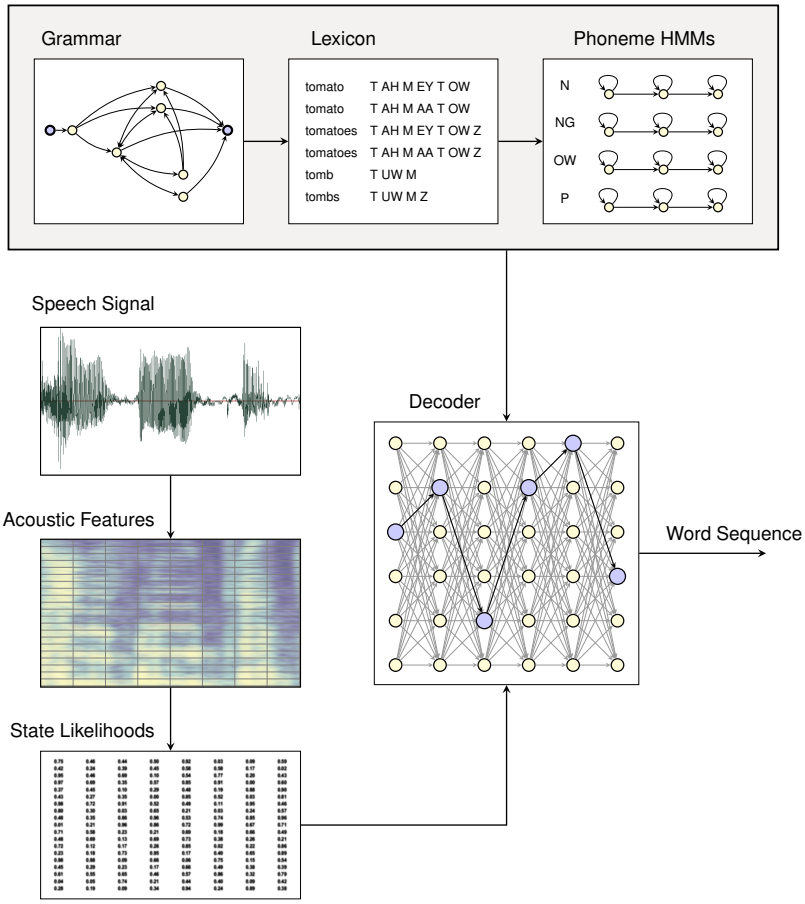


Figure 2.3. Block diagram of the HMM speech recognition pipeline.

speech processing systems, use features that are based on mel-frequency cepstral coefficients (MFCCs). Although the details such as the feature dimensionality vary, MFCC features were used in this thesis with both Aalto ASR and Kaldi systems, so the general idea is given below.

A natural candidate for characterizing speech is its frequency content. Instead of describing the frequency spectrum directly, MFCC features are based on a related concept named *cepstrum*. Cepstrum characterizes the frequency content of the spectrum itself. Somewhat different definitions of the cepstrum exist, but the idea is to consider the logarithmic frequency spectrum as a signal, and perform another Fourier-related transform. For MFCC features, only a small number of coefficients is wanted, so a filter bank is applied on the power spectrum. The filters are spaced equally in the mel scale of frequency. Discrete cosine transform is taken from a sequence of logarithmic filter energies to obtain the final coefficients [21, p. 359].

By matching the observed features to the emission probability distributions, the state observation likelihoods of each HMM state are obtained for each frame. The structure of the search space comes from three different sources:

- the individual phoneme models that often consist of only three states each,
- a pronunciation dictionary, often called a lexicon, that maps words to phonemes, and
- a language model or a grammar that defines the allowed word sequences and their probabilities.

The component of a speech recognizer that is responsible for finding the best possible word sequence given the model and the observations is called a decoder. A decoder can expand the search space dynamically during recognition, or the search network can be compiled from finite-state transducers (FSTs) before the recognition begins. The advantage of the dynamic search network is smaller memory consumption, while decoding is very fast when the network is compiled in advance.

The search space contains all the possible sequences of words from the vocabulary used by the decoder. Unless the grammar is restricted to a certain set of sentences, the search space is extremely large. No matter what kind of decoder is used, it is impossible to perform an exhaustive search. Instead, some heuristics are used to drop unlikely hypotheses at an early stage. The most important heuristic is *beam pruning*—at any given time, hypotheses whose probability is not close enough to the best hypothesis, are pruned out.

The speech recognizers used in this thesis use the HMM framework. In Publications I and II the AaltoASR system was used. In Publications III and IV, both AaltoASR and the Kaldi speech recognizer [74] were used. Most of the experiments used Gaussian mixture models (GMMs) for the emission probabilities. The Estonian NNLM experiments in Publication III and the experiments in Publication V used Kaldi with deep neural network (DNN) emission probabilities. AaltoASR uses a token pass decoder that extends the search network dynamically [38, p. 726]. Kaldi uses a search network that is compiled from four FSTs: HMM (phoneme model structure),

lexicon, grammar, and an FST that translates context dependent phonemes to context independent phonemes.

2.3 N-best Lists and Word Lattices

It is often useful to obtain multiple hypotheses from a decoder, instead of just the best one. As speech recognizers keep a list of the most likely hypotheses in memory while decoding, such an *n-best* list can be produced with no additional cost. When a long utterance is recognized, a list that includes sufficient alternatives for all the words can become extremely large. A word lattice is a more compact representation of the best hypotheses.

A word lattice is a a graph, where recognized words are saved as nodes or links. At any point of time, the decoder expands the current partial hypotheses and performs beam pruning. Any new words that are kept after pruning are saved in the word lattice. Usually the lattice also incorporates the alignment and acoustic and language model probability of the individual words. The decoding beam controls the size of the generated lattice.

Saving multiple hypotheses allows reordering them quickly afterwards using new language models, and with models that would be too expensive to use during the actual speech recognition. It also enables keyword spotting as a postprocessing step, brings in new possibilities for the estimation of confidence on the recognition result, and allows interactive applications to display multiple choices for the user. N-best lists or word lattices are also regularly used in discriminative training methods, as described in the following sections.

2.4 Training HMM-based Acoustic Models

The parameters of an HMM model include the transition and emission probabilities. Training an HMM-based acoustic model involves learning the parameters for all speech sounds. Traditionally this has been based on maximizing the likelihood of the parameters, given the observed acoustic features $\{\mathbf{o}_t\}$, when the reference transcription $\mathbf{w}$ is known. In other words, a certain HMM structure is defined by $\mathbf{w}$, and the objective is to find parameters θ that maximize the probability of the acoustic observations, $p(\{\mathbf{o}_t\} \mid \mathbf{w}, \theta)$.

Analytical solutions to the problem are not known, but the Baum-Welch algorithm is an efficient numerical method for finding the maximum-likelihood (ML) parameters. It is based on a general concept called expectation maximization (EM) [58]. Knowing the latent states $\{z_t\}$ and assuming the form of the emission probability distribution, it would be possible to compute the emission and transition probabilities that maximize the probability of the acoustic features. Because the latent states are not known, the algorithm operates on the expectation of the likelihood with respect to the conditional distribution of the latent variables given the training data.

The algorithm iterates expectation (E) and maximization (M) steps. The E step computes statistics of the states under the current estimates of the HMM parameters, and the M step calculates the parameter values that maximize the expectation assuming the computed statistics. Sufficient statistics for finding the parameters that maximize the expectation are $\{p_t(z \mid \mathbf{o})\}$, the probability distribution of the states at each audio frame, and $\{p_t(z, z' \mid \mathbf{o})\}$, the joint probability distribution of being in state z at frame t and in state z' at frame $t+1$. These are called the *state occupancies*.

In order to compute the statistics, Baum-Welch uses the forward-backward procedure [5, p. 168]. A *forward probability* is defined as the probability of observing a specific sequence of acoustic frames until time t, and ending up in state z. A *backward probability* is defined as the probability of being in state z at time t and observing a specific sequence of acoustic frames starting from time $t+1$. The algorithm is an example of *dynamic programming*. The forward and backward probabilities are computed iteratively for every state at every time step. Then the state occupancies can be expressed using the forward and backward probabilities.

Given enough training data and assuming that our model is able to represent the real probability distribution $p(\{\mathbf{o}_t\} \mid \mathbf{w})$, a decoder that is based on the maximum-likelihood solution would make optimal decisions [68]. Iterating the E and M steps improves the likelihood, but does not necessarily converge to the global optimum. Even if the global optimum is found, an HMM is an approximation that can never model speech perfectly. The accuracy of the model is also limited by the amount of available training data.

For the above reasons, even though the maximum-likelihood model is often close to optimal, better models for practical speech recognition tasks can be found by discriminative training. The idea is to maximize the discrimination between the correct transcription and other hypotheses,

instead of just maximizing the probability of the correct transcription. A maximum-likelihood model is still used as a starting point for discriminative training.

Several different discriminative criteria have been proposed. One popular choice is maximum mutual information (MMI) estimation. It attempts to make the mutual information between two random events, the reference transcription w and the observations $\{\mathbf{o}_t\}$, as large as possible [4]. This is equal to normalizing the ML objective by the probability of the observed signal, which in turn is a sum over all possible word sequences: $p(\{\mathbf{o}_t\}) = \sum_{\mathbf{w}} p(\{\mathbf{o}_t\} \mid \mathbf{w})p(\mathbf{w})$

The MMI criterion attempts to increase the likelihood of the correct transcription, but simultaneously decrease the likelihood of all possible word sequences. Clearly evaluating all the possible word sequences is intractable in continuous speech recognition with large vocabularies. In practice it is enough to operate on word lattices that incorporate the most likely hypotheses of each training utterance [95]. The lattices are generated once and then used during several training iterations.

Most of the acoustic models in this thesis were trained using the maximum-likelihood principle. Several Kaldi models in Publication III and the English Kaldi models in Publication IV were refined using the discriminative MMI criterion.

2.5 Pronunciation Modeling

A pronunciation dictionary defines the pronunciation of every word that an HMM-based speech recognizer is able to recognize. More specifically, it maps each word to a sequence of phoneme HMMs. The way in which a spoken language is written is called an *orthography* of the language. Finnish orthography is phonemic, letters more or less corresponding to the International Phonetic Alphabet (IPA). With regard to automatic speech recognition, this feature of Finnish language makes it easy to create the pronunciation dictionary using simple rules. Other languages may not be written in as systematic way. In particular, the pronunciation of most letters in English text varies from word to word. English speech recognizers use pronunciation dictionaries that are at least partly created by human experts.

2.5.1 Modeling Pronunciation Variation in a Dictionary

Language evolves and words are pronounced differently in different dialects. Pronunciation also often changes in conversations, and consecutive words tend to fuse together to make speech more fluent. Often multiple pronunciations are defined in the pronunciation dictionary for some words, such as the English word *tomato*. However, modeling pronunciation variation that depends on the word context can be challenging. One approach is to define alternative pronunciations for *multiwords*, sequences of a few words that are frequently pronounced differently together [25]. These pronunciations can be added manually, or some automatic method can be developed to generate new variants. Different prior probabilities can be given for the variants.

Pronunciation variant probabilities can be computed from the relative frequencies of the pronunciation variants in phonetic transcripts. Handwritten phonetic transcripts are rarely available for large corpora, but they can be obtained by aligning word-level transcripts using the Viterbi algorithm. In Publication I, multiword probabilities were computed after first generating phonetic transcripts of the training data using a speech recognizer. It is possible to use multiwords without changes in the decoder by estimating a language model from text where the corresponding word sequences have been substituted by multiwords. However, it is more accurate to train the language model on single words and split multiwords in the decoder [25]. In Publication I the decoder was modified to split multiwords into individual words before computing language model probabilities.

It might be tempting to include as much colloquial pronunciation variants, and pronunciations used in different dialects, as possible. The problem is that the chance of confusion between the words is increased by adding new pronunciations. Especially if pronunciations are added by some automatic method, some of the new pronunciations increase confusability without improving the recognition. Publication I presents a novel method that can be used to prune harmful pronunciations from a dictionary with a lot of pronunciation variants.

The pronunciation pruning method was inspired by discriminative methods for training acoustic models. It operates on word lattices that are generated from all training utterances. While word lattices usually include just one instance of a word that represents all its pronunciation variants (or the most likely variant), for pruning we generated lattices that

include a separate instance for all pronunciation variants. This allowed us to compute exactly the effect on recognition accuracy from removing a pronunciation variant. Initial tests were carried out in Publication I on an English multiword dictionary that was created by phoneticians, showing only small improvement. Later in Publication III the method was tested for pruning automatically generated foreign name pronunciations. Pronunciation pruning reduced recognition errors in the foreign names in Finnish broadcast news recordings by 2 %.

2.5.2 Pronunciation Variation in Finnish Written Conversations

There is a lot of variation in how colloquial Finnish is spoken. Because the orthography is very systematic, these changes are often written as well, which provides an interesting insight into colloquial speech. Phonological processes such as elision (*miksi* → *miks*) and internal sandhi (*menenpä* → *menempä*) can be visible in written colloquial Finnish. There is not a single correct way to transcribe a colloquial word. The same word can be condensed in multiple ways, depending on how informal the speaking style is, and how clearly the speaker wants to pronounce the word in the particular occasion. Furthermore, it is not always easy to tell the exact phonetic form of a spoken colloquial word, as the sounds may be somewhere in between two phonemes. Here are 20 ways to say the word "ninety" in Finnish, all of which also appear in written form on the Internet: *yhdeksänkymmentä*, *yhdeksänkymment*, *yheksänkymmentä*, *yheksänkymment*, *yhdeksäkymmentä*, *yheksäkymmentä*, *yheksäkymment*, *yhdeksänkytä*, *yhdeksänkyt*, *yhdeksäkytä*, *yhdeksäkyt*, *yheksäkytä*, *yheksäkyt*, *yheksänkytä*, *yheksänkyt*, *yhdekskyt*, *yhekskytä*, *yhekskyt*, *yheskytä*, *yheskyt*.

The relaxed colloquial forms are often ambiguous. In particular, grammatical information, that is normally conveyed by inflection, may be lost. Below are some example sentences, first in standard Finnish, then one or more ways in which they can be pronounced or written in informal conversations:

- *maalata kattoa* → *maalata kattoo* (to paint a/the roof)

- *porata kattoon* → *porata kattoo* (to drill into a/the roof)

- *katsoa* → *kattoa*/*kattoo* (to watch)

- *hän katsoo* → *se kattoo* (he/she watches)

- *he katsovat* → *ne kattovat/katsovat/kattoo* (they watch)

- *mennä katsomaan* → *mennä kattomaan/katsoo/kattoo* (to go to watch)

Different word forms are used in different situations and by different speakers. When the context does not give enough clues about the exact meaning of an ambiguous word, the speaker has to use a word that is closer to the standard form. For example, in relaxed speech, *minä menen* (I [will] go) can be elided to *mä meen* or even *mä mee*. Sometimes the personal pronoun is omitted, as the inflection can express the person; for example, one can answer a question simply by saying *meen*. Using *mee* alone could be confusing, however, as that is most often used to mean the imperative form *mene*.

The same phonetic form being used for many different meanings is obviously challenging for modeling the language. In the example above, the same word form *kattoo* is used for two inflections of the word *katto* (a roof) and four inflections of the word *katsoa* (to watch). Adding *kattoo* as an alternative pronunciation for the six word forms would increase confusion between the words. Especially so because the pronunciation probabilities are independent of the context.

Finnish dictionaries with alternative colloquial pronunciations are not available. An enormous amount of work would be required to create such a dictionary considering all the pronunciation variation in conversational Finnish. Because different pronunciations are expressed as different word forms in Finnish conversations, different pronunciation forms could in theory be found by clustering words in written conversations using automatic methods. Then harmful pronunciations could be pruned out using the discriminative method presented in Publication I. We use an alternative strategy in this thesis. Different pronunciation are modeled as different words in the language model. Each word form has just one pronunciation, which can be created automatically using rules based on the Finnish orthography.

There is a subtle difference between modeling the pronunciation variation in the language model and modeling it in the dictionary: The pronunciation probabilities in a dictionary are independent of the context, while a language model uses the context as a cue for predicting the next word

form. Thus using the language model to predict pronunciations should in principle reduce the confusability between words. The downside is that this increases data sparseness—there are even less examples of different sequences of word forms. This issue is discussed in the next section.

2.6 Data Sparseness in Finnish Conversations

Vocabulary size influences the difficulty of language modeling in many ways, so we wanted to compare the vocabulary size in standard and conversational Finnish texts in Publication I. On one hand, the different ways in which words are written in Finnish conversations increase the vocabulary. On the other hand, we generally tend to use a simpler vocabulary when having a conversation, and many words are reduced to short ambiguous forms. We showed that on the whole, Finnish online conversations used a larger set of different word forms than the the same amount of formal speech.

A very large vocabulary is challenging both from the perspective of computational efficiency and model performance. Traditional n-gram language models that are based on statistics on how frequently a sequence of words is observed in the training data regard words as separate entities, meaning that they cannot generalize what they learn from one word to other similar words. In the following example, the word *luentoa* appears in two context that are semantically very similar, even though the word forms are different:

- *mullon maanantaina kaks luentoa*

- *mulla on tiistaina kolme luentoa*

The first sentence means “I have two lectures on Monday”, and the second sentence means “I have three lectures on Tuesday”. Their colloquial pronunciations have been written down phonetically. There are actually a large number of different ways in which these sentences could be written. Our intuition says that the probability of the word *luentoa* (lectures) in one context is most likely similar to its probability in the other, but traditional n-gram language models do not see any similarity between these contexts. We say that the data is *sparse*, because a lot of training data is needed to model the word in every possible context.

In this thesis it is shown that two language modeling techniques that generalize better to unseen contexts are useful in modeling conversational Finnish. Class-based language models, presented in Section 3.3, can be used assuming we have a method for grouping similar words together. Neural network language models, presented in Chapter 4 automatically learn a mapping of words to continuous-valued vectors, with semantically similar words close to each other in the vector space.

Another prominent feature of Finnish language with regard to automatic speech recognition is agglutination, which further increases the number of different word forms and data sparsity. Most words actually consists of smaller units, called morphs, that bear some meaning. Subword models, presented in Section 3.4, model language as a sequence of units that are shorter than word. In a sense they can be seen as a solution to the same problem as class-based models—both reduce data sparsity by decreasing the vocabulary size. Publication V compares these approaches.

2.7 Evaluating Speech Recognition Performance

Measuring the performance of a speech recognizer is very well established. The aim is simply to produce as few errors as possible. The standard error measure is the word error rate, defined as the minimum number of word substitutions (N_s), deletions (N_d), and insertions (N_i) that are needed to correct the result, relative to the total number of words in the reference transcription (N_w):

$$WER = \frac{N_s + N_d + N_i}{N_w} \tag{2.1}$$

To put this into perspective, 10 % WER can be considered very good for an automatic speech recognizer, while a value larger than 100 % means that more edit operations are needed to correct the result, than would be needed to write the text starting from scratch. The performance of an automatic speech recognizer can also be compared to that of a human transcriber. The accuracy of a transcription created by a human obviously depends on many factors, including how difficult the speech is to recognize, whether the transcriber is allowed to listen to the audio repeatedly, and how experienced the transcriber is. Earlier studies have found the accuracy of a nonprofessional transcriber on read English speech (Wall Street Journal corpus) to be around 2 % [24] and the accuracy of a professional transcriber on spontaneous English to be anywhere between 4 % and 11 % depending

on the corpus and how carefully the speech is transcribed [96].

It can be argued that for agglutinative languages WER is too inaccurate. A related measure called letter error rate (LER) or character error rate (CER) has also been commonly used in for example broadcast news tasks that contain standard Finnish. Perhaps the most typical error is an incorrectly recognized suffix. Such error would increase WER by $1/N_w$. On contrast, LER would be increased by the ratio of incorrect characters to the total number of characters in the text, meaning that LER penalizes less for errors that cause only a few letters to be incorrectly recognized, compared to errors that cause all or most of the letters of a word to be incorrectly recognized. There are several reasons why WER is still used to evaluate the Finnish tasks as well throughout this thesis:

- WER has become the standard measure for assessing speech recognition accuracy, while LER is mostly used among Finnish researchers. Thus WER is more meaningful to most researchers.

- The purpose of LER is to penalize more for completely incorrect words, but there is rarely confusion between words that are not phonetically similar (in which case they are for the most part written using the same characters).

- Recognizing suffixes correctly is important for the understandability of text, so this thesis aims to develop good language models that can help the recognizer to select the grammatically correct word forms from other phonetically similar forms. Thus one may argue that incorrectly inflected words should be penalized as much as incorrect words.

- Recognizing a few characters incorrectly may not just cause the inflection to change, but can easily change the meaning of the word to something completely different. In this case the error is as severe as if all the characters are recognized incorrectly.

- While incorrect inflections should count as errors, pronunciation variation should not. Alternative word forms were included in the reference transcripts for different pronunciations. This is explained below. It would be possible to define alternative word forms and compute the minimum LER, although it might be confusing that recognizing a word incorrectly

would be penalized less if the word has many alternative pronunciations.

As discussed in the previous section, conversational Finnish vocabulary contains many similar surface forms of the same word, not just because of inflection, but also because the different pronunciations are expressed in written words. Comparing the recognition result to the phonetically exact transcription is not suitable for evaluating a speech-to-text task. The error rate would fluctuate as many alternative pronunciations would have very similar probabilities. Even for a human transcriber, it is sometimes difficult to tell which phonemes the speaker used. This problem can be solved by defining alternatives for each word in the transcripts, and computing the minimum WER considering the alternative ways to write the sentence.

In all the conversational Finnish experiments in this thesis, evaluation set transcripts included alternations for different pronunciation variants. The alternations also included compound words written separately. Defining the alternations was a laborious task, since the acceptable ways to write a word depend on its meaning in the particular sentence, and on the other words in the sentence, i.e. the context needed to be considered when adding the alternations. Those were systematically created for the development set only in Publication V. The reference transcripts with alternations have been released with the DSPCON corpus.

Regardless of the challenges discussed above, measuring accuracy of speech recognizer output is straightforward. However, the performance of a speech recognizer depends on many factors such as the decoding parameters, pronunciation dictionary, and how well the acoustic and language models perform together. When developing language models, it would be convenient to evaluate the language model alone, disregarding all the other factors. Section 3.6 discusses how language models can be evaluated independently.

3. Statistical Language Models

3.1 Probabilistic Model of Language

A statistical language model can be used to evaluate how likely a sentence would occur in a given language. These probabilities are estimated from a large text corpus. Language models are useful in many applications, such as machine translation, information retrieval, and handwriting recognition. In speech recognition, a language model assigns a prior probability to every word sequence, which is combined with acoustic likelihoods to evaluate the probabilities of different hypotheses.

In some applications a speech recognizer can be restricted to a set of sentences defined by a grammar. In others a statistical language model is trained that should be able to assign a probability to every possible word sequence—even those that do not occur in the training corpus. The most important property of a statistical language model is the ability to generalize to unseen sentences.

The vast majority of statistical language models are based on factorizing the probability distribution of a word sequence into the product of word conditional probabilities using the chain rule of probability:

$$p(w_1, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_1, \ldots, w_{t-1}) \tag{3.1}$$

The word conditional probabilities on the right side of Equation 3.1 easily become too low to be representable using double-precision (64-bit) floating-point numbers. The standard way of solving this problem is to compute probabilities in log space. This means that instead of a product, a sum of log probabilities is computed:

$$\log p(w_1, \ldots, w_T) = \sum_{t=1}^{T} \log p(w_t \mid w_1, \ldots, w_{t-1}) \tag{3.2}$$

Traditionally the model is simplified for computational reasons by making the *Markov assumption*, i.e. assuming that the probability of the next word depends on only a fixed number of previous words. Such models are called n-gram language models, because they can be estimated using only n-gram[1] statistics. The rest of this chapter presents variations of n-gram language models that are estimated using interpolation or back-off techniques. Chapter 4 is devoted to neural network language models.

3.2 N-gram Language Models

N-gram models have dominated language modeling for several decades. They are based on the assumption that only the previous $n-1$ words can influence the probability of any word. The notation $\mathbf{h}_t^n$ will be used to denote the preceding n words at time step t. Equation 3.1 is replaced with an approximation where word probabilities are conditioned only on $\mathbf{h}_t^{n-1}$:

$$p(w_1, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid \mathbf{h}_t^{n-1}) \tag{3.3}$$

Generally models that are based on such an assumption are called $(n-1)^{\text{th}}$ order Markov models.

N-gram language models are still used in most of the speech recognizers during the first recognition pass, because they are simple and very efficient to use. In a speech recognizer, the search space is greatly reduced, because all the partial hypotheses that end in the same $n-1$ words can be recombined by dropping all but the best hypothesis. When using more advanced language models, typically decoding is done in multiple passes. The first pass is performed using an n-gram model, quickly evaluating numerous hypotheses. The best hypotheses are saved in a word lattice, which can be decoded using more complex language models. In all the speech recognition experiments in the publications of this thesis, n-gram models were used during the first pass.

The maximum likelihood estimate for $p(w \mid \mathbf{h}^{n-1})$ is the relative frequency of word w in the context $\mathbf{h}^{n-1}$:

[1] An n-gram is simply a sequence of n words.

$$p_{ML}(w \mid \mathbf{h}^{n-1}) = \frac{c(\mathbf{h}^{n-1}, w)}{\sum_{w'} c(\mathbf{h}^{n-1}, w')}, \tag{3.4}$$

where $c(\mathbf{w})$ is the count of n-gram $\mathbf{w}$ in the training data. The probability in Equation 3.4 is nonzero for n-grams that occur in the training data. Those estimates are the parameters of the model. Because the number of parameters is not fixed, the model is said to be nonparametric.

Different variations of the n-gram model have been proposed that modify Equation 3.4 to generalize to unseen n-grams. They work generally by reducing the probability of those n-grams that occur in the training data, and distributing the discounted probability mass to unseen n-grams, essentially *smoothing* the probability distributions. Usually these methods combine information from n-grams of different lengths by backing off or interpolation.

3.2.1 Smoothing

A *back-off* model [45] discounts the probability of the n-grams $\left(\mathbf{h}^{n-1}, w\right)$ that occur in the training data by coefficient $d(\mathbf{h}^{n-1}, w)$ and distributes the discounted probability mass to n-grams of length $n-1$. The probability of unseen n-grams is defined recursively:

$$p_{BO}(w \mid \mathbf{h}^{n-1}) = \begin{cases} d(\mathbf{h}^{n-1}, w)\, p_{ML}(w \mid \mathbf{h}^{n-1}), & \textit{seen n-grams} \\ \beta(\mathbf{h}^{n-1})\, p_{BO}(w \mid \mathbf{h}^{n-2}), & \textit{unseen n-grams} \end{cases} \tag{3.5}$$

The back-off weights $\beta(\mathbf{h}^{n-1})$ are chosen to normalize the conditional probability distribution.

Models of different order can also be combined by *interpolation*. A recursive definition is often used:

$$\begin{aligned} p_I(w \mid \mathbf{h}^{n-1}) = {} & \lambda(\mathbf{h}^{n-1})\, p_{ML}(w \mid \mathbf{h}^{n-1}) \\ & + \left(1 - \lambda(\mathbf{h}^{n-1})\right) p_I(w \mid \mathbf{h}^{n-2}) \end{aligned} \tag{3.6}$$

The interpolation weights $\lambda(\mathbf{h}^{n-1})$ can be optimized on a held-out set, but the optimization requires some consideration. We expect a weight to be higher when the corresponding maximum likelihood estimate $p_{ML}(w \mid \mathbf{h}^{n-1})$ is more reliable, i.e. when the context $\mathbf{h}^{n-1}$ occurs frequently in the training data. [43]

Many smoothing methods can be seen as a variation of the back-off or interpolation scheme described above. Kneser–Ney smoothing [50] has

been particularly successful. The most important novelty is that for any order $k < n$, the n-gram counts in the maximum likelihood estimate are replaced by so-called *continuation counts*:

$$p_{continuation}(w \mid \mathbf{h}^{k-1}) = \frac{N_{1+}(\cdot, \mathbf{h}^{k-1}, w)}{\sum_{w'} N_{1+}(\cdot, \mathbf{h}^{k-1}, w')} \tag{3.7}$$

$N_{1+}(\cdot, \mathbf{w})$ is the number of different words that precede the n-gram $\mathbf{w}$. When $k > 1$, the continuation probability in Equation 3.7 is also smoothed. Kneser and Ney used absolute discounting: instead of discounting the maximum likelihood estimate by a factor $d(\mathbf{h}^{n-1}, w)$ as in Equation 3.5 or $\lambda(\mathbf{h}^{n-1})$ in Equation 3.6, they subtracted a fixed number from the numerator of Equation 3.7. The intuitive explanation why Kneser–Ney smoothing works well is that when there are only few different words in the training data that precede $\mathbf{w}$, and w' is not one of them, we should give $\mathbf{w}$ a low probability after w'. On the other hand, if $\mathbf{w}$ follows many different words in the training data, it is more likely that it will also appear in a new context.

Chen and Goodman proposed modified Kneser–Ney smoothing [13], which has been used throughout this thesis. It combines the estimators of different order using an interpolation scheme that is a bit different from Equation 3.6: first the higher-order estimate is discounted and then a weighted lower-order estimate is added. Additionally, while the original Kneser–Ney smoothing discounted a fixed number, the modified version discounts a different number depending on the count itself.

3.2.2 Maximum Entropy Models

The previous section discussed backing off and interpolation for combining n-gram models of different order. A third technique is worth mentioning, even though it was not used in this thesis. *Maximum entropy* is a theoretically well justified principle for combining statistical knowledge. In the context of language modeling, the knowledge is usually n-gram statistics, for example the n-gram counts up to a given maximum order. The idea is to select a probability distribution that is consistent with the statistics, and otherwise as uniform as possible.

Maximum entropy language models [9] seek to maximize the entropy of

the target word conditional on the context:[2]

$$\mathcal{H}(W \mid \mathbf{H}^{n-1}) = -\sum_{\mathbf{h}^{n-1},w} p(\mathbf{h}^{n-1}, w) \log p(w \mid \mathbf{h}^{n-1}) \tag{3.8}$$

The conditional entropy in Equation 3.8 measures the uncertainty in the output, when the context is known. Maximizing it is equivalent to selecting as uniform output distribution as possible.

The constraints are expressed using *feature functions* $\{f_i(\mathbf{h}^{n-1}, w)\}$, binary indicators that select particular n-grams. The constraints impose that the expectation of the feature probability must match the training data. The solution to the constrained optimization problem can be expressed using Lagrange multipliers $\{\lambda_i\}$, and has the form of a log-linear model:

$$p_{ME}(w \mid \mathbf{h}^{n-1}) = \frac{\exp(\sum_i \lambda_i f_i(\mathbf{h}^{n-1}, w))}{\sum_{w'} \exp(\sum_i \lambda_i f_i(\mathbf{h}^{n-1}, w'))}, \tag{3.9}$$

It can be shown that the maximum entropy model is actually the log-linear model that maximizes the likelihood of the training data [9, p. 9].

3.3 Class-based Language Models

In Finnish and other highly agglutinative languages, the number of different word forms that appear in a corpus is huge, and most word forms are used very rarely. In conversational Finnish the phonological processes that transform words are also visible in written form. As discussed in Section 2.6, it becomes very difficult to reliably estimate the probability of the rare word forms in new contexts. This can be helped by grouping words into classes and estimating probabilities for class n-grams.

The most common formulation of a class based language model assumes that every word belongs to exactly one class. We denote $c(w)$ the function that maps words to classes. The probability of a word within its class can be made dependent on the history, but usually it is estimated as the relative frequency of the word within the class, regardless of the history [11]:

$$p(w \mid \mathbf{h}^{n-1}) = p(c(w) \mid c(\mathbf{h}^{n-1}))\, p(w|c(w)), \tag{3.10}$$

[2] In practice, the reasonable assumption is made that the empirical probability from the training data can be used as the marginal probability of the context: $p(\mathbf{h}^{n-1}, w) = p_{ML}(\mathbf{h}^{n-1})\, p(w \mid \mathbf{h}^{n-1})$
This modification avoids normalization of model probabilities over the possible contexts [76, p. 13].

where $c(\mathbf{h}^{n-1})$ denotes a sequence of the previous $n-1$ classes.

In the previous section we assumed that the basic unit that is modeled is a word, but $p(c(w) \mid c(\mathbf{h}^{n-1}))$ can be modeled using all the mentioned techniques easily by simply replacing words with the corresponding class in the training data. Both n-gram and neural network models based on word classes were used in Publications IV and V. Section 3.8 provides an overview of how the word classes were created in those publications.

3.4 Subword Language Models

A language model estimates the probability of any sentence, usually as the product of the conditional probabilities of the words in that sentence. Subword language models simply use the product of the conditional probabilities of some smaller units than words. This is very straightforward in principle, but before such models can be used for speech recognition, one has to determine 1) how the subword units are derived, 2) how word boundaries are represented, and 3) how the pronunciations of the subwords are derived.

Because Finnish orthography is phonemic, subword pronunciations can be generated using the same rules that are used to generate word pronunciations. In English speech recognition, subword units have been used mostly in recent end-to-end systems. As described in Section 2.1, such systems use a single DNN that outputs letters, without modeling the phoneme acoustics, language, and word pronunciations separately.

A natural candidate for subword units in synthetic languages is *morphs*, the smallest grammatical units in a language. Words can be segmented to morphs using a morphological analyzer or using statistical methods [36]. Both approaches can work well, but morphological analyzers recognize only a certain vocabulary, and there are no morphological analyzers that recognize colloquial Finnish word forms.

3.4.1 Morfessor

Morfessor is a family of unsupervised methods for splitting words into smaller fragments [18]. The Baseline version of Morfessor was used for creating subwords in Publications III and V and is described here. The resulting units resemble linguistic morphs, and are often called *statistical morphs* or just morphs. Two implementations of the method [17, 94]

have been published.[3] Statistical morphs have been successfully used in speech recognition of many agglutinative languages [52, 15]. Compared to word models that were limited to typical vocabulary sizes of the time, subword models based on statistical morphs improved speech recognition considerably.

The objective is to learn a subword unigram model from a corpus. The assumption that the subword probabilities are independent of the context is clearly false, but seems to work in practice and enables an efficient search algorithm [94, pp. 13–16]. The model parameters θ define the vocabulary of subword units that in the rest of this discussion are called morphs, and the unigram probabilities of the morphs. There are generally multiple ways in which a corpus can be segmented into units defined by the morph vocabulary. During model training, a particular segmentation is assumed for each word. After the model is learned, it is possible to find a better segmentation using the same morph vocabulary.

The cost function that Morfessor defines is inspired by the minimum description length (MDL) principle [77]. The cost can be interpreted as the minimum number of bits required to encode the model and the data using the model [84]. However, a more recent interpretation formulates the task as maximum-a-posteriori (MAP) estimation of the parameters θ. The posterior probability of the parameters given the training data is $p(\theta \mid \mathbf{w})$, which using the Bayes' theorem and by taking the logarithm turns into the following cost function:

$$\mathcal{C}(\theta, \mathbf{w}) = -\log p(\theta) - \log p(\mathbf{w} \mid \theta) \tag{3.11}$$

The Bayesian interpretation of probability allows treating θ as a random variable and defining the prior distribution $p(\theta)$ to express our uncertainty on the model parameters without any additional knowledge.

The likelihood of the parameters θ is the probability of a segmented training corpus $\mathbf{w}$ given the parameters $p(\mathbf{w} \mid \theta)$. It can be computed as the product of the maximum-likelihood unigram probabilities, which are the relative frequencies of the morphs in the segmented training data.

The prior can be derived by assuming the following production model for the training corpus. There is a given probability distribution over the letters of the alphabet, which includes a morph boundary symbol. The size of the morph vocabulary $|V|$ is drawn randomly, and then enough

[3] The latest version of the Morfessor tool is available on GitHub:
`https://github.com/aalto-speech/morfessor`

letters are generated for such a vocabulary. Then the total token count of the corpus is drawn randomly, and this is distributed to the morphs to obtain the individual morph counts $c(m_i)$. Finally, the corpus is generated randomly according to those counts.

From the above assumptions, noninformative priors are derived for the probabilities of the morph strings and their counts [94, p. 7]. The relative frequency of a letter in the training data is taken as the probability of the letter. The probability of a morph strings is the product of the probabilities of the letters in the string. This is multiplied by $|V|!$ to obtain the probability for an unordered vocabulary. Probabilities for the morph counts $c(m_i)$ are obtained using a binomial coefficient, noting that each distribution of the total token count $\sum c(m_i)$ to $|V|$ individual morphs is equally likely:

$$p(c(m_1) \dots c(m_{|V|})) = 1 / \binom{\sum c(m_i) - 1}{|V| - 1} \tag{3.12}$$

One way to look at MAP training is as regularization of maximum-likelihood training. Generally splitting words into sequences of smaller units does not increase the likelihood.[4] The likelihood increases when a morph vocabulary is chosen that is close to the word vocabulary. The prior probability increases when a vocabulary is chosen that contains fewer and shorter morphs.

Intuitively, minimizing both terms in Equation 3.11 should produce a concise subword vocabulary that can model the training data well. Minimizing just the first term would produce a small vocabulary that cannot represent the data efficiently, whereas minimizing the second term would result in a large vocabulary that can represent the corpus using few tokens. It may be useful to add a hyperparameter α to control the balance between these two objectives [51]:

$$\mathcal{C}(\theta, \mathbf{w}) = -\log p(\theta) - \alpha \log p(\mathbf{w} \mid \theta) \tag{3.13}$$

The standard training algorithm uses a greedy search that splits words and subwords recursively. At any given time, the model defines the segmentation of each word. The algorithm starts by each word forming a single morph. At its simplest, the algorithm processes one morph at a time.

[4] This is easy to understand by thinking about the set of all possible sentences, which will share the total probability mass. A word vocabulary can be used to model sentences that contain only words from the vocabulary, resulting in highest likelihood. Morphs can represent a larger set of sentences than words. Letters can represent all the sentences that can be written using the alphabet, distributing the probability mass to an even larger set of sentences.

All possible ways to split the morph into two are considered along with the possibility of not splitting it. The one that minimizes the cost is selected. [94, pp. 13–15]

In the end there are multiple ways to segment the corpus using the created morph vocabulary. The probability of different segmentations can be computed using the morph probabilities given by the model. After the training finishes, the best segmentation for each word can be found using a modified Viterbi algorithm. In this formulation, the hidden state sequence consists of morphs, and the observations are letters. Thus one hidden state corresponds to multiple observations. [94, pp. 11–12]

3.4.2 Maximum-Likelihood Models

As mentioned in the previous section, the maximum-likelihood word model has higher likelihood than subword models, and further splitting subwords generally does not increase the likelihood. Morfessor uses the prior probability of the model parameters to regularize the maximum-likelihood training. An alternative is to optimize the likelihood of a unigram model, but use some heuristics to split words into smaller units. A simple search strategy that optimizes the likelihood alone iterates segmenting the training text using a subword vocabulary, and creating new subwords by resegmenting a word randomly if it meets specific criteria [16].

A different model altogether was used in two experiments in Publication III. It models text as a sequence of latent variables called *multigrams* that give rise to variable-length letter sequences up to some maximum length [22]. Only the concatenation of the letter sequences is observed. The model can be used to infer the boundaries of the multigrams, for example word boundaries from concatenated words, or subword boundaries from words. Generally it is possible for the same multigram to emit different letter sequences in order to model e.g. corrupted text [22, p. 230]. When using multigrams to model subwords, we assumed that the same multigram always emits the same letter sequence.

The parameters of the model are the prior probabilities of the multigrams. The algorithms that are used for estimation of the parameters and segmentation of the corpus are similar to those that are used to do inference on hidden Markov models (HMMs), but they are adjusted to work with a sequence of variable-length observations—they process text letter by letter, but compute probabilities for multigrams.

Segmenting text means finding the most likely multigram sequence,

given the current model parameters. A Viterbi search is used in the same way as Morfessor segments words after training. The maximum-likelihood parameters can be estimated using expectation maximization (EM). The EM algorithm iterates by re-estimating the parameters so that they maximize the expectation of the likelihood, with regard to the probability distribution of the different segmentations under the current estimates of the parameters. A forward–backward procedure can be used to compute the probability of a multigram (in this case a subword) occurring at given position, which gives an efficient parameter re-estimation formula [22, p. 227]. An approximation is also possible by iterating Viterbi segmentation and re-estimating the parameters as the relative frequencies of the subwords.

The Greedy 1-grams (G1G) [92] strategy was used to segment text in two experiments in Publication III. Target size of the vocabulary is set explicitly. G1G starts from a large subword vocabulary. First the EM algorithm is used to find probabilities for the subwords, and an iterative process is used to remove low-probability subwords. Then a finer pruning is done by removing subwords that decrease the likelihood least, until a desired vocabulary size is reached. The change in likelihood is estimated by segmenting the text using Viterbi with and without the subword in the vocabulary. This approach has worked well with large corpora and reasonably large subword vocabulary sizes.

3.5 Combining Multiple N-gram Language Models

When a single large corpus is not available for the target language, combining data from multiple sources becomes an issue. Simply concatenating all the data may not give the best result, if the corpora differ in size and relevance. One possibility is to train a separate model from each corpus and interpolate probabilities from the component models, leading to the following model:

$$p(w \mid \mathbf{h}^{n-1}) = \sum_j \lambda_j \, p_j(w \mid \mathbf{h}^{n-1}) \tag{3.14}$$

Interpolating the probabilities during inference from multiple models is slower and complicates the system. A more practical approach that was used in Publication II, III, IV, and V is to combine the component models into a single model whose probabilities approximate model interpolation

[89]. All n-grams that can be found in any of the component models are assigned a probability that is a weighted average of the original probabilities. If the n-gram does not exist in all the component models, some component probabilities may be computed by backing off. Then the back-off weights of the final model, $\beta(\mathbf{h}^{n-1})$ in Equation 3.5, are recomputed to normalize the model.

The mixture weights can be optimized on the development data using an expectation-maximization (EM) algorithm [58, pp. 13–16]. The algorithm was used to derive weights for the mixture models in this thesis, except for the subword models in Publication V, where we found hand-tuning necessary. The algorithm can be derived from the assumption that each word is generated by just one of the component models, indicated by the random variable Z. The mixture weights are interpreted as the probability that a random word is generated by the corresponding component, $\lambda_i = p(Z = i)$. This is similar to how the HMM parameters were estimated by iteratively computing the state occupancies in Section 2.4.

The latent variables $\{\mathbf{z}_t\}$ are vectors that indicate the components that generated the words $\{w_t\}$. If word w_t was generated by the mixture component i, $z_{ti} = 1$ for the i^{th} element of the vector. Knowing the values of the vectors, the maximum-likelihood estimates for the mixture weights could be computed as the proportion of words generated by the corresponding component:

$$\hat{\lambda}_i = \frac{1}{T} \sum_t z_{ti}, \tag{3.15}$$

where T is the total number of words. However, in the absence of those values, the algorithm maximizes the expectation of the likelihood, with respect to the conditional distribution of the latent variables given the development data.

In the expectation (E) step of the algorithm, sufficient statistics are computed for the expectation under the current estimates of the weights $\{\lambda_i\}$. For each word w_t in the context $\mathbf{h}^{n-1}$, the probability that the word was generated by component i is denoted τ_{ti}:

$$\begin{aligned} \tau_{ti} &= p(Z_t = i \mid w_t, \mathbf{h}_t^{n-1}) \\ &= \frac{p(Z_t = i, w_t \mid \mathbf{h}_t^{n-1})\, p(\mathbf{h}_t^{n-1})}{p(w_t \mid \mathbf{h}_t^{n-1})\, p(\mathbf{h}_t^{n-1})} \\ &= \frac{\lambda_i\, p_i(w_t \mid \mathbf{h}_t^{n-1})}{\sum_j \lambda_j\, p_j(w_t \mid \mathbf{h}_t^{n-1})} \end{aligned} \tag{3.16}$$

Now the expectation of the complete-data (including the latent variables) log likelihood for the new mixture weights $\{\lambda_i^*\}$ can be expressed using $\{\tau_{ti}\}$:

$$\begin{aligned}
&\sum_t \mathrm{E}\big[\log \mathcal{L}(\{\lambda_i^*\} \mid \mathbf{z_t}, w_t)\big] \\
=&\sum_t \sum_i \tau_{ti} \log p(z_{ti}, w_t \mid \mathbf{h}_t^{n-1}, \lambda_i^*) \\
=&\sum_t \sum_i \tau_{ti} \left[\log p(z_{ti} \mid \lambda_i^*) + \log p(w_t \mid \mathbf{h}_t^{n-1})\right]
\end{aligned} \tag{3.17}$$

In the maximization (M) step, the new mixture weights are selected to maximize the expectation in Equation 3.17. Only the left term depends on the mixture weights, leaving the following function to be maximized:

$$\sum_t \sum_j \tau_{ti} \log p(z_{ti} \mid \lambda_i^*) = \sum_t \sum_j \tau_{ti} \log \lambda_i^* \tag{3.18}$$

The maximum-likelihood estimates can be computed as in Equation 3.15, but averaging the membership probabilities τ_{ti} instead of the latent indicator variables z_{ti}.

By iterating the E and M steps, the algorithm converges to a solution that may not be the global optimum, but most of the time works reasonably well.

3.6 Evaluating Language Models

Cross entropy is often used to evaluate how well a statistical model predicts the outcome of a random phenomenon. When evaluating a language model, a word sequence $\mathbf{w} = w_1, \ldots, w_T$ is seen as T samples from some random phenomenon whose probability distribution is not known. Empirical cross entropy of the observed word sequence is defined to approximate the cross entropy between the unknown distribution and the word conditional probabilities $p(w_t \mid w_1, \ldots, w_{t-1})$ predicted by the language model (see Equation 3.1):

$$\mathcal{H}(\mathbf{w}, p) = -\frac{1}{T} \sum_{t=1}^{T} \log p(w_t \mid w_1, \ldots, w_{t-1}) = -\frac{1}{T} \log p(\mathbf{w}) \tag{3.19}$$

The standard measure of language model performance is however (empirical) *perplexity*, which is used in the evaluations throughout this thesis. It is defined as the exponent of empirical cross entropy:

$$PP(\mathbf{w}, p) = \exp(\mathcal{H}(\mathbf{w}, p)) = \exp(-\frac{1}{T}\log(p(\mathbf{w}))) = \frac{1}{p(\mathbf{w})}^{\frac{1}{T}} \tag{3.20}$$

The lower the perplexity value, the better the model fits the data.

Equation 3.20 does not define what happens when the word sequence contains unknown words, i.e. when $p(w_t \mid w_1, \ldots, w_{t-1})$ is not defined. There are two approaches that have commonly been used:

- **Closed vocabulary**. Unknown words are simply ignored, meaning that they are not taken into account when computing $p(\mathbf{w})$, and they are not included in the word count T.

- **Open vocabulary**. An unknown word token (typically <unk>) is included in the vocabulary and determines the probability of all words that are outside the vocabulary. The probability can be defined as a constant that is discounted from the observed unigram probabilities, or it can be estimated by replacing all words in the training data that are outside the vocabulary with the <unk> token.

This choice is not important with regard to speech recognition, as a decoder is only able to hypothesize words that exist in its language model. Closed-vocabulary language models can be compared as long as the vocabulary is the same, because then every model excludes the same words from the perplexity computation. However, the question of how to handle out-of-vocabulary (OOV) words became essential in Publications II and III, in the context of text filtering. The problem was that perplexity was used to evaluate different text segments against the same language model, instead of different language models against the same text.

A closed-vocabulary language model ignores the OOV words, which are the words that occur with low probability. A text segment that contains many OOV words gets a low perplexity. If a filtering method selects low-perplexity text segments, it actually ends up selecting segments with many low-probability words.

The problem with an open vocabulary is that determining the probability of the <unk> token is very difficult. A common approach is to replace all OOV words in the training data with <unk> before training. This obviously means that all the words from the training data cannot be included in the model. The vocabulary is normally constructed by excluding words that occur only a few times. The probability of <unk> is actually the probability

of the set of excluded words, so it depends on how many words are excluded from the vocabulary. There is nothing to say that this results in a good probability estimate for a single word.

In the conversational Finnish data that was collected for this thesis, the vocabulary is so large, that most vocabulary words occur only once. In Publications II and III, we noticed that excluding the words that occur only once from the vocabulary would already result in a too high estimate for the <unk> probability. The solution was to use subword models, introduced in Section 3.4. The choice of subword units is not as crucial for filtering as it is for speech recognition. In these publications, words were segmented using the Morfessor Baseline algorithm.

3.7 Variable-Order and Pruned N-gram Models

The number of possible n-grams grows exponentially with the model order. Of course, all the possible n-grams do not exist in the training data. With more training data there will be more n-grams that can be used to estimate the model. Some method is usually applied to prune out the least relevant n-grams, in order to reduce the model size. A simple but effective method is to exclude those n-grams that appear only very few times in the training data. Often the minimum n-gram counts are set higher for higher-order n-grams. For example, in most publications in this thesis, we used a minimum count of 2 for 4-grams.

Another strategy is to remove an n-gram from the model, if the difference to backing off to the lower-order probability estimate is small enough. Since the back-off weights have to be recomputed after pruning, this approach can be improved by comparing the pruned model p' to the original model p using relative entropy, also known as Kullback–Leibler divergence [88]. The conditional relative entropy of two conditional probability distributions, $p(w \mid \mathbf{h}^{n-1})$ and $p'(w \mid \mathbf{h}^{n-1})$, is defined

$$D_{\mathrm{KL}}(p \parallel p') = \sum_{\mathbf{h}^{n-1},w} p(\mathbf{h}^{n-1}, w) \log \frac{p(w \mid \mathbf{h}^{n-1})}{p'(w \mid \mathbf{h}^{n-1})}. \tag{3.21}$$

Minimizing the relative entropy can also be seen as minimizing the increase of the perplexity of the model [88]. The perplexity of a model is defined as the exponent of entropy. While the empirical perplexity in Equation 3.20 was used to evaluate a model on the empirical distribution of text, the perplexity of a model can be seen as evaluating the model on

the original model distribution:

$$PP(p) = \exp(\mathcal{H}(p)) = \exp\Big(-\sum_{\mathbf{h}^{n-1},w} p(\mathbf{h}^{n-1}, w) \log p(w \mid \mathbf{h}^{n-1})\Big), \quad (3.22)$$

where the definition of entropy $\mathcal{H}(p)$ is for a conditional probability distribution $p(w \mid \mathbf{h}^{n-1})$ and the sum is over the n-grams of the model. In Publication II, in the text filtering experiments in Publication III, and in Publication IV, n-grams whose removal caused only a small relative increase in the model perplexity were pruned out. The threshold was in the order of 10^{-9}.

Typically word n-gram models that are used in speech recognition are not estimated on longer than 4-grams, because of diminished benefits and more computation required by using higher-order models. Significantly longer contexts are sometimes beneficial when modeling language using subword units. The smaller the subword units, the longer context is needed to model text well. At one extreme the subword vocabulary includes just the letters of the alphabet. The vocabulary consist of a few dozen letters, and the length of a sentence is the number of letters in the sentence. At the other extreme, the vocabulary contains whole words. The vocabulary can consist of millions of words, and the length of a sentence is the number of words in the sentence.

Even if the subword vocabulary is smaller than a word vocabulary, building a model of very high order first and then pruning it may be too costly. An alternative is to grow the model incrementally, only adding the most relevant n-grams. The Kneser–Ney growing algorithm starts from a unigram model and increases the model order by one until no new n-grams are added [85]. On each iteration, the algorithm processes the longest n-grams in the model. Taking one of those n-grams from the model, $\mathbf{w}_i$, it finds all the extensions $G = \{\mathbf{w}_i, w_j\}$ from the training data. The set of n-grams G is added to the model, if adding it reduces the cost.

The cost function is based on the MDL principle [77], sharing some similarity with the Morfessor cost. It consists of two parts, the log probability of the training data and the model size. Similar to Morfessor, a hyperparameter is added to control the relative importance of these two objectives. However, Kneser–Ney growing uses it to weight the model size term, while the Morfessor α by convention weights the log probability of the training data (see Equation 3.13). Kneser–Ney growing was used in Publications III and V.

3.8 Forming Word Classes

Finding good word classes is not easy. The approaches can be divided into two categories: rule-based methods that use prior knowledge of which words have a similar function in the language, and unsupervised methods that are based on co-occurrence of words in the training data.

Prior knowledge of the language may include for example part-of-speech information. It is always language specific and often difficult to obtain. The intuition is that when we group together words that are used in similar contexts, the probability estimate for a class in a given context represents well the probability of the individual words in that class. Unfortunately the relationship between words in any natural language is so complex that linguistic information does not necessarily correspond to a classification that is good from the perspective of modeling a word sequence using Equation 3.10. The suitability of different word clustering methods for language modeling is compared in Publication V.

3.8.1 Unsupervised Methods

Several unsupervised methods have been used for clustering words into some meaningful classes. The algorithms that have been used for class-based language models are mostly based on optimizing the perplexity of a class bigram model. This is equivalent to using the following objective function, which is the log probability of the training data:

$$\mathcal{L} = \sum_t [\log P(c(w_t) \mid c(w_{t-1})) + \log P(w_t \mid c(w_t))] \tag{3.23}$$

Maximizing the class bigram likelihood in Equation 3.23 is a very hard problem because of the number of different ways in which N_C classes can be formed from N_V words. In practice the clustering algorithms that try to maximize this objective are greedy—they use some heuristic to guide the search in such a way that it may stop in a local optimum.

The exchange algorithm [49] starts with some initial clustering of N_C classes. The initial clustering can be based for example on word frequencies in the training data. The algorithm progresses by moving one word at a time to another class. It is evaluated how much the objective would change if the word was moved to each of the N_C classes. If any move would improve the objective, the word is moved to the class that would produce the biggest improvement.

Variations of this algorithm have been implemented in the mkcls tool [70]. An efficient implementation of the algorithm maintains counts of class–class, word–class, class–words, and word–word bigrams, which makes it possible to evaluate the change in the objective for a single class in $\mathcal{O}(N_C)$ time [57]. Thus one word can be processed in $\mathcal{O}(N_C^2)$ time. The algorithm can be continued until there are no more moves that would improve the objective. However, usually the algorithm converges much sooner to such a state that it can be stopped without sacrificing much of the quality of the resulting classes. The algorithm can be parallelized by evaluating the change in objective for different classes in different threads.[5] In Publication IV, word classes were created using mkcls. In Publication V, a faster multi-threaded exchange implementation was used.[6]

The Brown clustering method [11] uses the same objective, but a different search strategy. It starts with each word in a distinct class and merges classes until there are only N_C classes. The algorithm needs to evaluate the change in objective after merging a pair of classes. By caching some statistics, the evaluation can be done in a time that is proportional to the number of classes that appear together with the merged class in the training data. Assuming that this is a very small number, a single evaluation can be thought of running in constant time. This can still be too slow if the algorithm starts with N_V classes—evaluating all pairs of classes would take $\mathcal{O}(N_V^2)$ time. The authors propose an approximation to further reduce the computational complexity: The algorithm starts with N_C most frequent words in distinct classes and at each iteration adds a new word to one of those classes. Using this strategy, the running time of one iteration is $\mathcal{O}(N_C^2)$. The algorithm stops after $N_V - N_C$ iterations.

Neural networks provide a different kind of method for clustering words. Section 4.1 shows how the projection layer of a neural network language model maps words to a continuous vector space. The word embeddings that an NNLM learns tend to capture syntactic and semantic regularities of the language [64]. Word classes can be formed by clustering these vectors using traditional clustering techniques.

Word embeddings can be created using very simple network architectures, such as the continuous bag-of-words (CBOW) and continuous skip-gram

[5] Parallelizing the computation of the objective for example on a GPU is difficult, because the word–class and class–word bigram statistics are sparse and cannot be saved in a regular matrix.

[6] The implementation of the exchange algorithm, developed at Aalto University, is available on GitHub: `https://github.com/aalto-speech/exchange`

(CSG) [60]. These architectures do not contain a nonlinear hidden layer. The CBOW model predicts the current word from the average of the embeddings of the context words. The CSG model predicts the context words (irrespective of order) given the current word. The context includes words immediately before and after the current word.

The computational cost of training the CBOW and CSG models is dominated by the output layer. The training can be made very fast using softmax approximations that are summarized in Section 4.2. After training the model, the word embeddings are clustered for example using the k-means algorithm [55]. The cost of one iteration of k-means has linear dependency on the number of classes and the dimensionality of the embeddings, but the number of iterations required for convergence depends on the data. Publication V shows that this method does not work well for class n-gram models. The method does not explicitly optimize language model performance, and clustering high-dimensional data is known to be difficult.

3.8.2 Rules for Clustering Conversational Finnish Words

It would seem plausible that linguistic knowledge on the similarity of words could be used to estimate models that generalize better to unseen n-grams. As discussed in Section 2.5.2, the same Finnish word can be written in various different ways, depending on the degree to which the conversation is colloquial. In order to accurately predict words using a class-based model, we would like to have the words in each class to be similar in the sense that they could be used interchangeably. While the probability of a word may depend on the degree to which the text is colloquial, it is reasonable to assume that the model should give a similar probability regardless of which form of the same word is used.

By observing how different phonological processes alter the letters in a word, it is possible to create rules that recognize different variants of the same word. Publication V presented a rule-based method for clustering colloquial variants of Finnish words. First two vocabularies are compiled, one from a standard Finnish and one from a colloquial Finnish corpus. The algorithm starts by assigning each word to a separate class. Then it compares every standard Finnish word to every colloquial word. If the colloquial word appears to be a variant of the standard Finnish word according to the rules that describe typical letter changes, those two classes are merged.

Because the reduced word forms may be ambiguous, eventually multiple different standard Finnish words may get into the same class, but this is rare. Typically only a handful of word forms will be in each class. In Publication V this rule-based method was combined with the exchange algorithm to create larger classes. This clustering method gave the best word error rate, but the difference to other methods was very small.

3.9 Details on the N-gram Models Used in This Thesis

In all the publications in this thesis, the first speech recognition pass was performed with an n-gram language model. The models were trained using the SRILM toolkit [89]. Except for the class-based models, the modified Kneser–Ney smoothing was used. Modified Kneser–Ney computes the discount parameters using statistics on how many n-grams occur one, two, three, and four times. The estimation fails if any of these *count-of-counts* are zero. Class-based n-gram models were used in Publications IV and V. When training a language model on word classes, those count-of-counts are often zero. For example, there are very rarely class unigrams that occur only once. Thus the class-based models were estimated using interpolation of the maximum likelihood probabilities, as in Equation 3.6.

As there was not a single high-quality corpus of conversational Finnish, the n-gram models were combined from different sources of varying size and quality. A single model with interpolated probabilities was created, optimizing the interpolation weights on development data using the EM algorithm. The number of submodels used in the interpolation varied between publications.

In Publication II five submodels were trained from different web data sets. In Publication III, the conversational Finnish models were interpolated from six web data sets and conversational Estonian from four. In Publication IV, the Finnish models were interpolated from six web data sets and DSPCON corpus. The English models were interpolated from ICSI and Fisher corpora and one web data set. Unsurprisingly, an improvement was seen over pooling all the data together. However, in Publication IV, because of lack of data, we had no separate evaluation set for Finnish after optimizing the interpolation weights.

While writing Publication V, we noticed that optimizing the interpolating weights for seven submodels tends to overlearn the development data. Better evaluation results were achieved by using just two submodels for

Finnish: DSPCON corpus and web data. For Estonian, three submodels were used, separating a large set of broadcast transcripts from a smaller set of spontaneous conversations.

The class-based models in the publications were based on Equation 3.10. In Publication IV, Finnish speech recognition results from a class-based 4-gram model were given for reference. It was on the same level with a word model trained on a 200,000-word vocabulary, and interpolating these two models gave a small improvement. Different methods for clustering words into classes and different class vocabulary sizes were compared in Publication V. We found, surprisingly, that n-gram models trained on word classes consistently outperformed full-vocabulary word models, even without interpolation.

There are several differences between the class models in Publications IV and V. In Publication IV, mkcls implementation of the exchange clustering algorithm and Aalto ASR decoder were used. In Publication V, an exchange implementation developed at Aalto University was used, and Kaldi decoder with DNN acoustic models that provided a huge leap in the baseline performance. Perhaps the most important difference is, however, that the vocabulary used to create the word lattices was limited to 200,000 words. It seems that class-based modeling is beneficial especially when using a very large vocabulary.

First results from subword models on conversational Finnish ASR were given in Publication II. The subword vocabularies were created using the Morfessor Baseline algorithm and their sizes were in the order of 100,000 morphs. The results were clearly worse than results from a slightly smaller word vocabulary.

In several experiments in Publication III, Morfessor Baseline was used to create segmentations from several Finnish and Estonian data sets. The data was from broadcast news, books, and other sources that do not contain much conversational language, and comparison to word models was not included in these experiments.

In Publication V, Morfessor Baseline models were used again on the conversational Finnish task. The hyperparameter α was used to control the size of the resulting morph vocabulary. The vocabulary turned out very large, more than 400,000 words, with $\alpha = 1.0$, which worked well in class-based subword models, but otherwise lower α-values worked better. The subword n-gram models were variable-order models trained using the VariKN toolkit. The variable-order subword models still did not outperform

word models, but neural network subword models did. Neural network language models will be presented in the next chapter.

The usability of subword models for speech recognition may be limited in some languages by the fact that a pronunciation cannot be defined for units shorter than a word. Commonly an HMM decoder needs a pronunciation for all the lexical items. In Publication III, a different kind of decoder was constructed that is based on subword units, but limits the search space to words that are defined in a pronunciation dictionary. In this experiment, Morfessor was used to create subwords from a small broadcast news data set, but from a larger set of the same data, subwords were created to maximize the likelihood of the multigram model using the G1G search.

In Publication III, the multigram model was also used in the web data selection experiments. Subword probabilities were estimated using the EM algorithm and the Viterbi algorithm was used to segment the downloaded conversations given those probabilities. The subword model included individual letters, meaning that any sentence could be segmented.

Kneser–Ney growing was used to build the subword n-gram models in Publications III and V. The resulting models were 6-gram, 8-gram, and 12-gram models.

4. Neural Network Language Models

4.1 Artificial Neural Networks

Artificial neural networks have been successfully applied to various machine learning applications from stock market prediction to self-driving cars. Today all the state-of-the-art speech recognizers also used neural networks for modeling the acoustics and the language. A neural network is a structure of nested mathematical operations. As a whole, the network can approximate complex nonlinear functions, for example probability distributions. Each node of the network, called a *neuron* or a *unit*, performs some operation on its inputs. The neurons are organized in layers. The inputs of a neuron are either data samples or outputs of other neurons. Typically a neuron uses the outputs of the neurons in the layer immediately before it to compute its own output, but different network topologies can be defined.

Figure 4.1 shows a network that consists of three fully-connected layers. By convention, the inputs (blue circles) are drawn at the bottom of the graph. Next are two hidden *hidden layers*. Each neuron computes its output using its inputs and a set of parameter variables. It is convenient to represent the output of a layer, $\mathbf{s}^{(l)}$, and the input of the network, $\mathbf{x}$, as vectors. The output of a hidden layer is called the *hidden state* of that layer. The network can be seen as a chain of vector-valued layer functions: $\mathbf{y} = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$.

A basic feedforward layer function is an affine transformation followed by a nonlinear *activation function*: $f(\mathbf{x}) = \phi(W\mathbf{x} + \mathbf{b})$ The activation function $\phi(\mathbf{z})$ is typically the hyperbolic tangent or the rectifier ($\max(0, x)$) applied elementwise. The input of the activation function, $\mathbf{a} = W\mathbf{x} + \mathbf{b}$, is called the *preactivation* vector or the *net input* of the layer. The weight matrix W and the bias vector $\mathbf{b}$ are parameters of the layer that will be learned from

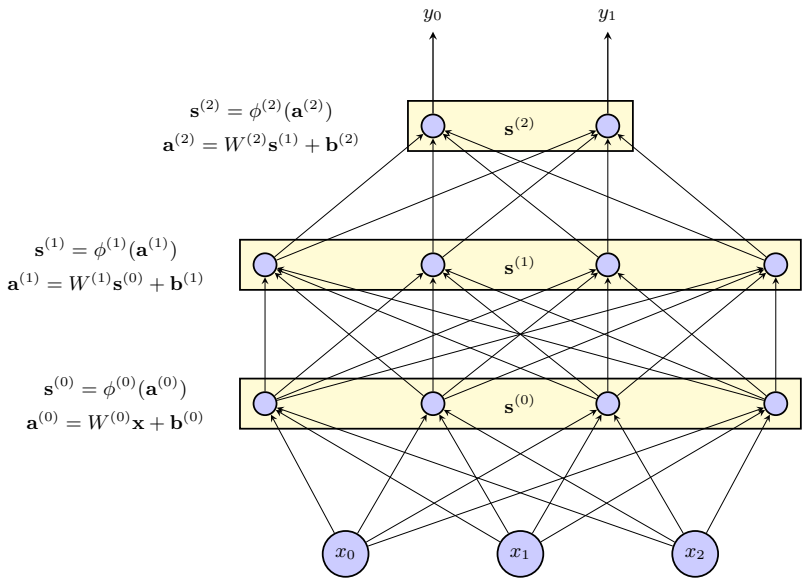


Figure 4.1. An artificial neural network with two hidden layers and an output layer. The output of the final layer (y_i) can approximate probability distributions that depend on a large number of inputs (x_i).

the training data.

In a classification task, the activation of the final layer is usually softmax:

$$y_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}, \tag{4.1}$$

where a_i is the i^{th} preactivation of the softmax layer. The outputs, y_i, estimate the probability of each class. This is equivalent to Equation 3.9, meaning that softmax is actually a log-linear model. However, the features in maximum entropy models have been fixed when designing the model, whereas the first layer of a neural network language model automatically learns to represent the input words as continuous-valued feature vectors.

In language modeling, a neural network can be used to predict a word occurring in given context. Perhaps the most straightforward approach is a *feedforward* network that is essentially an n-gram model—the output is the probability of word w_t and the input is the $n-1$ previous words. The interesting novelty is how words are used as input. Bengio et al. [6] reasoned that words should be mapped to continuous vectors in order for the neural network to learn semantic similarities between words that in the vocabulary are represented by discrete numbers. This mapping is done by the projection layer in Figure 4.2.

The projection matrix is a parameter of the network, and will be learned

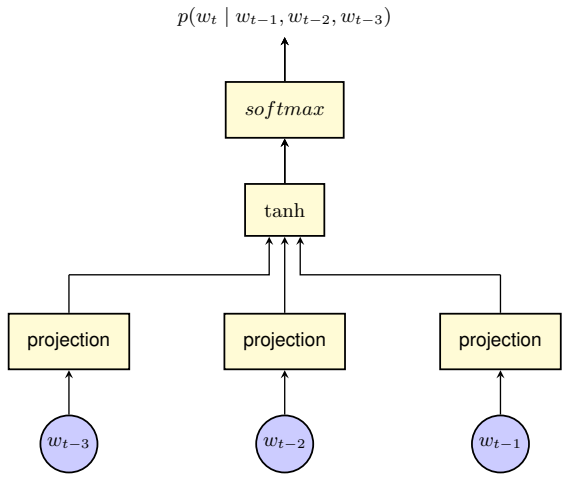


Figure 4.2. A feedforward neural network language model. The projection layer maps words into continuous vectors. There is just one projection matrix that is applied to all the inputs.

during training just like the weights of the other layers. The projection can be implemented by selecting the row from the matrix that corresponds to the index of the word in the vocabulary. Another way to look at it is as a multiplication between the projection matrix and a vector that contains one at the location specified by the word index and zeros elsewhere (*1-of-N encoding*). All the inputs are projected using the same matrix.

The projection layer maps words into vectors, sometimes called *word embeddings* or *word features*. The dimensionality of the projection is typically between one hundred and a few hundred. Feedforward networks have a fixed context length that is typically below 10. As the number of inputs grows, so does the number of network parameters. For example, if a 5-gram feedforward NNLM (4 input words) uses 100-dimensional word embeddings, the next layer has 400 inputs.

If the network contains cycles, it is called a recurrent neural network (RNN). Typically the cycles connect neurons within the same layer. One way to think of recurrent connections is that the hidden state of a layer is transferred to the next time step. The next time step computes its own output from its own input, but all time steps share the same network parameters. It was shown above, how a feedforward network can be used to model a word sequence (or generally a time series) by predicting the next word given a fixed number of previous words. RNNs can predict the next word given the previous word and a state that is retained between time steps. The state is designed to capture the dynamics in the sequence up to

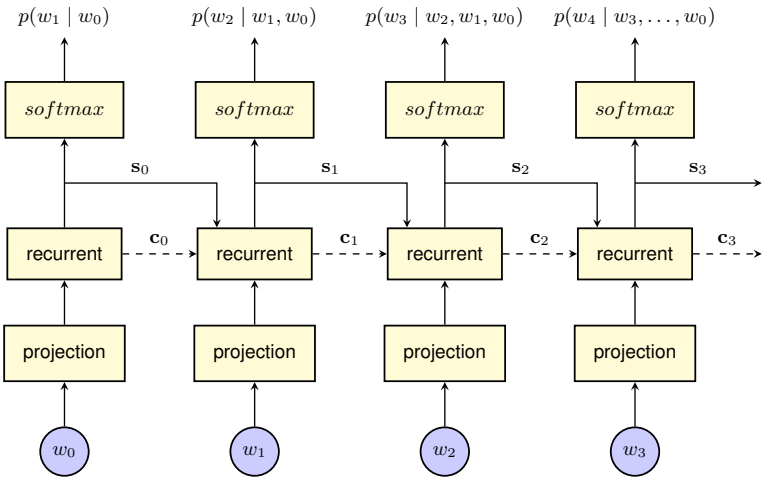


Figure 4.3. An RNN language model. The recurrent network has been unrolled four time steps. The output of the recurrent layer, $\mathbf{s}_t$ is passed to the next time step. LSTM adds the cell state $\mathbf{c}_t$, which is designed to convey information over extended time intervals. The same network parameters are used at every time step.

the current time step. Consecutive words are fed as input at consecutive time steps, and each time step updates the state.

RNNs can be implemented by unrolling the recurrency as shown in Figure 4.3. Each time step uses a copy of the same network, but different input. Each time step also outputs the probabilities for the next time step. The difference to a feedforward network is that the recurrent layer output (the hidden state) at each time step is used also as its input at the next time step.

4.2 Suitability of Neural Networks for Language Modeling

Language modeling is a form of sequence prediction. The unique feature that language models have is that their input and output space is discrete and very large, consisting of hundreds of thousands of words. This is a classification problem with a large number of separate classes, but the classes are not unrelated. A neural network can learn a mapping from words to vectors of a continuous space, where the vector dimensions bear some meaning.

As we noticed in Publications IV and V, and as various other authors have noticed before [61, 2, 14] neural network language models can clearly exceed the accuracy of n-gram models in terms of prediction accuracy. Al-

though in principle the RNN state can capture long-term dependencies, when the duration of the dependencies increases, learning the dependencies becomes difficult in practice [8]. Learning long-term dependencies with RNNs is discussed in Section 4.4.

The most important disadvantage of neural networks compared to n-gram models is the computational complexity. Training an NNLM can take weeks, and in fact the performance of an NNLM is to a large extent limited by the time that we are willing to spend on training the model. Also the inference is slower, which makes them unsuitable for one-pass speech recognition. The larger the vocabulary is, the more difficult it becomes to implement a language model with neural networks. This is because the softmax normalization in Equation 4.1 takes time linear in vocabulary size.

Therefore it should not be surprising that a lot of the research on NNLMs has concentrated on reducing the computational complexity of the output layer. These approaches can be divided into four categories:

- **Smaller vocabulary**. The neural network is trained on a different vocabulary, either word classes, subwords, or a shortlist of the most frequent words.

- **Approximate softmax**. The softmax is approximated by performing the normalization only on a random subset of the vocabulary on each training item. This speeds up training, but usually normal softmax is taken to correctly normalize the model during inference.

- **Hierarchical model**. The output layer is structured into a hierarchy of multiple softmax functions, each taken over a small set of classes. This speeds up both training and inference.

- **Unnormalized model**. The training objective is altered to guide the training toward a model that is approximately normalized. If exact probabilities are not needed, softmax is not necessary and inference is fast.

In the neural network language modeling experiments in this thesis, an n-gram language model was used to create a word lattice in the first speech recognition pass, and the lattice was rescored using an NNLM in a second

pass. Rescoring a lattice does not require as fast evaluation of neural network probabilities, so unnormalized models were not used. Training time is still a severe issue especially in languages such as Finnish that have a very large vocabulary.

By choosing a different vocabulary, both training and inference speed can be increased dramatically. Usually this means an approximation of a full-vocabulary model, which can decrease the model performance. The traditional approach is to use the neural network to predict just a small subset of the vocabulary, called a *shortlist*, and use a back-off n-gram model to predict the rest of the vocabulary. A shortlist is convenient to implement within a feedforward NNLM, which is a particular kind of an n-gram model itself.

Class-based NNLMs were used in Publications IV and V. Publication V also compared subword and shortlist NNLMs, softmax approximations, and hierarchical softmax. The softmax approximations are discussed in Section 4.5.

4.3 Training Neural Networks

So far we have discussed the different network architectures, but disregarded the most essential challenge in using artificial neural networks—how to learn such parameter values that the network models our data well. The learning algorithms that have been found to work best with neural networks are based on gradient descent, iterating computation of the gradient of the cost function with respect to the network parameters, and adjusting the parameters accordingly.

The cost function $\mathcal{C}(\theta, \mathbf{w})$ measures the difference between the actual and desired output of the network. The training data w provides the inputs and target outputs of the network, the word following the input words taken as the target output. Different ways to define the cost function in NNLM training are discussed in Section 4.5. Common to those definitions is that the cost, and consequently its gradient, can be written as a sum of subgradients taken over individual training examples:

$$\mathcal{C}(\theta, \mathbf{w}) = \sum_{w_t} \mathcal{C}(\theta, w_t) \tag{4.2}$$

$$\nabla_\theta \mathcal{C}(\theta, \mathbf{w}) = \sum_{w_t} \nabla_\theta \mathcal{C}(\theta, w_t), \tag{4.3}$$

The input of the network is needed for computing the cost, but has been omitted from the right side of Equations 4.2 and 4.3 for brevity. In a feedforward network the input would be a few words that precede the target word. In a recurrent network the input is the words from the beginning of the sentence up to the one before the target word. An NNLM can be designed to take dependencies between sentences into account by using a context that spans multiple sentences.

4.3.1 Stochastic Gradient Descent

The parameters of a neural network have to be optimized using a numerical method, usually stochastic gradient descent (SGD) or some variation of it. The objective is to minimize the value of the cost function $\mathcal{C}$ on the training data w. Gradient descent tries to achieve this by iteratively moving the model parameters θ in the direction where the cost decreases fastest. That direction is given by the negative gradient of the cost with respect to the parameters:

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta \mathcal{C}(\theta, \mathbf{w}) \tag{4.4}$$

The size of the step taken in the direction of the negative gradient can be adjusted by selecting a smaller or larger *learning rate* η. Selecting a good value is crucial for the convergence of the optimization. The smaller the value, the slower the training will be. On the other hand, a too large value will cause training to oscillate without ever converging. A common approach is to start with as large learning rate as possible and then decrease it when the model stops improving.

Computing the entire gradient $\nabla_\theta \mathcal{C}(\theta, \mathbf{w})$ would take a lot of time. Expressing the gradient as the sum of the subgradients at individual training examples, as in Equation 4.3, allows a stochastic version of gradient descent to be used. Instead of computing the whole gradient before updating the network parameters, the gradient is computed on a small subset of the training data called a *mini-batch*. The network parameters are updated after processing each mini-batch. Processing the entire training data is called an *epoch*. The order of the mini-batches is shuffled in the beginning of each epoch.

Updating the parameters after each mini-batch allows a more fine-grained tuning of the parameters and increases the performance of the final model. The batch size is an important training hyperparameter. A

too small batch size can make training unstable, causing the gradients to explode to infinity. The batch size also defines the amount of data that is processed at any instance. A too large batch size is problematic with regard to the memory consumption, and a too small batch size can be inefficient because there is not enough data that can be processed in parallel.

Many variations of SGD try to improve the speed of convergence. Nesterov's Accelerated Gradient avoids oscillation of the cost by updating the parameters according to a *momentum* vector that is slowly moved in the direction of the gradient [69]. Adagrad [23], Adadelta [97], Adam [46], and RMSProp adapt the magnitude of the subgradients (or equivalently, the learning rate) per parameter. The idea is to perform larger updates for those parameters that have been updated infrequently in the past. This is realized by keeping track of statistics of the past gradients for each parameter. The adaptive gradient methods do not require careful tuning of the learning rate hyperparameter. We found Adagrad to work especially well in different tasks and used it in Publications IV and V.

4.3.2 Backpropagation Algorithm

Computing the gradient involves a forward and a backward pass through the neural network. In a forward pass the network is fed with the input, and the cost function is computed as the output of the network. In a backward pass, the gradient of the cost function is computed with respect to all the parameters of the network iteratively for each layer, using the chain rule of differentiation. The procedure is called backpropagation.

The backpropagation starts by computing the gradient of the final layer with respect to the preactivations. For simplicity, let us now consider a network with a single output, and the mean squared error (MSE) cost function. MSE cost compares the network output y_t at time step t to the desired output y_t^*:

$$\mathcal{C}(\theta, w_t) = \frac{1}{2}(y_t - y_t^*)^2 \tag{4.5}$$

The gradient of the cost at time t with respect to all network parameters θ can be derived by recursive application of the chain rule. First the derivative of the cost with respect to the network output is computed. In case of the MSE cost, it is

$$\frac{\partial \mathcal{C}}{\partial y_t} = y_t - y_t^*. \tag{4.6}$$

Next the gradient of the cost with respect to the final layer preactivations $\mathbf{a}^{(out)}$ is expressed using the derivative in Equation 4.6 and the gradient of the network output. This vector quantity, $\mathbf{\Delta}^{(out)}$, is called the error in the output layer:

$$\mathbf{\Delta}^{(out)} = \nabla_{\mathbf{a}^{(out)}} \mathcal{C} = \frac{\partial \mathcal{C}}{\partial y_t} \nabla_{\mathbf{a}^{(out)}} y_t = (y_t - y_t^*) \phi^{(out)\prime}(\mathbf{a}^{(out)}) \tag{4.7}$$

In Equation 4.7 we have assumed that the output layer function is of the form $\phi^{(out)}(\mathbf{a}^{(out)})$. $\phi^{(out)}$ is a function whose derivative we know that is applied elementwise to the preactivations $\mathbf{a}^{(out)}$. For example, we know that the derivative of $\tanh$ activation is $1 - \tanh^2$, which is also taken elementwise. Because the values of $\mathbf{a}^{(out)}$ and y_t are known through forward propagation, we can compute the value of $\mathbf{\Delta}^{(out)}$ at the current input.

The error $\mathbf{\Delta}^{(l)}$ in layer l can be used to find the gradient with respect to the parameters of that layer. For example, assuming the input of the layer, $\mathbf{s}^{(l-1)}$, is transformed by the weight matrix $W^{(l)}$, we can get the gradient of the cost with respect to the weights (a matrix of the partial derivatives with respect to the elements of $W^{(l)}$) using

$$\left[\frac{\partial \mathcal{C}}{\partial w_{ij}}\right] = \left[\frac{\partial \mathcal{C}}{\partial a_i}\right] \odot \left[\frac{\partial a_i}{\partial w_{ij}}\right] = \mathbf{\Delta}^{(l)} \left[\mathbf{s}^{(l-1)}\right]^T, \tag{4.8}$$

where $\odot$ is the Hadamard or elementwise product. For brevity, the layer index l is left out from the weights and preactivations, i.e. $W^{(l)} = [w_{ij}]$ and $\mathbf{a}^{(l)} = [a_i]$. Again, the output of the previous layer, $\mathbf{s}^{(l-1)}$, is obtained by forward propagation.

The error in the output layer is also propagated down the network, in order to obtain the gradient with respect to the parameters of the previous layers. The gradient of the cost with respect to the previous layer output $\mathbf{s}^{(l-1)}$ is expressed using $\mathbf{\Delta}^{(l)}$:

$$\nabla_{\mathbf{s}^{(l-1)}} \mathcal{C} = \left[\frac{\partial a_j}{\partial x_i}\right] \nabla_{\mathbf{a}^{(l)}} \mathcal{C} = \left[\frac{\partial a_j}{\partial x_i}\right] \mathbf{\Delta}^{(l)} \tag{4.9}$$

The matrix in Equation 4.9 is the transpose of the Jacobian of the layer l preactivations. The preactivation vector is just an affine transformation of the input vector, meaning that the Jacobian can be derived easily. Without loss of generality, we will ignore the bias, so that the Jacobian is simply the weight matrix. The gradient can then be written $\left[W^{(l)}\right]^T \mathbf{\Delta}^{(l)}$.

This procedure would be continued by multiplying the gradient $\nabla_{\mathbf{s}^{(l-1)}} \mathcal{C}$ by the Jacobian of $\mathbf{s}^{(l-1)}$ to obtain the error in the previous layer. In the

case that the layer functions are just a linear transformation followed by an elementwise nonlinearity $\phi(\mathbf{a})$, we get a simple equation for the error in layer $l-1$ given the error in layer l:

$$\mathbf{\Delta}^{(l-1)} = \left[W^{(l)}\right]^T \mathbf{\Delta}^{(l)} \odot \phi^{(l-1)\prime}(\mathbf{a}^{(l-1)}) \tag{4.10}$$

The differentiation can be automated, requiring the user to specify only the forward computation as a graph of mathematical operations, each of which has a known derivative. The gradient with respect to any parameters can then be computed by performing the Jacobian-gradient multiplications for every operation in the graph. For details, see the book by Goodfellow et al. [28].

Recurrent networks are used to model time sequences. At each time step, a recurrent layer combines two inputs, the output of the previous layer and the hidden state of the previous time step. The output of the layer is written both to the next time step and the next layer. A simple way to extend backpropagation to recurrent networks is called backpropagation through time (BPTT).

BPTT is essentially backpropagation in a recurrent network that is unrolled in time as in Figure 4.3. Conceptually, the network is copied as many times as there are time steps, and each time step has its own input and output, but the network parameters are shared between all time steps. In practice, language modeling data sets are too large to perform backpropagation on a network that is unrolled for the whole length of the training data. Instead, mini-batches that contain only a small subset of the data points are processed independently.

Applying backpropagation in an unrolled recurrent network requires just well-known generalization of the chain rule to multiple inputs where the output of the layer branches. This is taken care by modern frameworks such as Caffe, Theano, and TensorFlow that perform automatic differentiation. Because the parameters are shared between the time steps, the same parameter tensors are used as input for all time steps. This will lead to the parameter updates accumulating from all time steps.

4.4 Learning Deep Representations

Deep neural networks (DNNs) can learn a hierarchical representation that models the data efficiently and generalizes to unseen data. For example, it has been shown that DNNs trained to classify images learn to recognize

more abstract concepts in the higher layers, using simpler forms that are recognized by the lower layers [98].

A recurrent neural network is a particular kind of a DNN. An RNN language model, such as the one in Figure 4.3, learns a structure that is good at predicting a word using information that is constructed from the previous words in a hierarchical manner. The first layer creates a vector representation or embedding of the word. A recurrent layer creates a hierarchical representation of the context. These can be followed by subsequent recurrent and feedforward layers.

The information that flows in an NNLM cannot be visualized in the same way as the activations of an image classifier. However, we can visualize relationships between word embeddings and observe semantic relationships in the vector space [64]. In the same way we can think of the activations of a recurrent layer at the end of a sentence as a *sentence embedding*. By visualizing the relationships between sentence embeddings we can observe that they capture semantic relationships between sentences [91]. Thus the recurrent state embodies some linguistic information about previous words and their relationship in the sentence.

While n-gram models typically use no more context than the previous three words, normally text contains also cues that can help prediction of words a much longer distance apart. Neural networks learn to represent these relationships using the backpropagation algorithm, which may be a limiting factor in how long relationships the network can learn.

Backpropagation essentially computes the effect that adjusting each parameter has on the cost function. Equation 4.10 describes how errors are backpropagated to the previous layer. It is important to notice what happens to the magnitude of the errors in deep networks. The derivative of typical activation functions such as the hyperbolic tangent and the sigmoid is between -1 and 1. The weights are also usually initialized to small values. Thus each layer reduces the magnitude of the errors until they are too small to be represented by the limited precision of floating point numbers [39, pp. 19–21]. This problem of *vanishing gradients* makes learning long-distance dependencies slow, if at all possible.

It would be possible to select an activation function that outputs larger values, but that would just introduce another problem of *exploding gradients*. Currently a popular choice for the activation function, at least in computer vision, is the rectifier, because its derivative is 0 for negative input and 1 for positive input. The rectifier reduces the problem because

its gradient does not vanish for large inputs.

There are also several other changes that have been proposed to deep networks, including RNNs, that make it easier for the networks to convey information over long distances. It should be noted that even though the network structure is changed, they address a problem in training networks. In principle an RNN could convey information over indefinitely long distances, if the parameters were chosen carefully, but backpropagation fails to learn such parameters.

Consider the RNN language model in Figure 4.3. At each time step, the recurrent layer uses two inputs, the hidden state from the previous time step, and the embedding of the current input word. Based on these it has to compute the state that it outputs to the next time step. In order to predict a word in the end of a sentence using information about which word appeared in the beginning of the sentence, all time steps in between have to learn to pass on some information in the hidden state that they read from the previous time step (rather than something they observed from the current input word). Furthermore, the error signal that controls this learning gets weaker the longer the distance in time is.

4.4.1 Long Short-Term Memory

To better facilitate information flow, a layer can offer a channel that passes information on without transforming it through an activation function. Long short-term memory (LSTM) [40] is a modification of the RNN structure that passes another state vector to the next time step, in addition to the hidden state. This *cell state* allows information to flow unchanged over long distances, because it is modified only by explicitly removing or adding information when necessary.

The key to controlling the cell state is gates that multiply a signal by a value that is most of the time close to either zero or one. The value is produced by taking the sigmoid of an affine transformation of the input. These control units are effectively layers themselves that learn to set some elements of a signal vector to zero. Each has its own set of weights that are learned during the network training.

LSTM uses gates to control when and how to modify the cell state. There are actually a number of variations around the same idea. The original paper used two gates: *input gate* decides what information is added to the cell state, and *output gate* decides what information is written to the hidden state. It is now standard to include a *forget gate* that clears

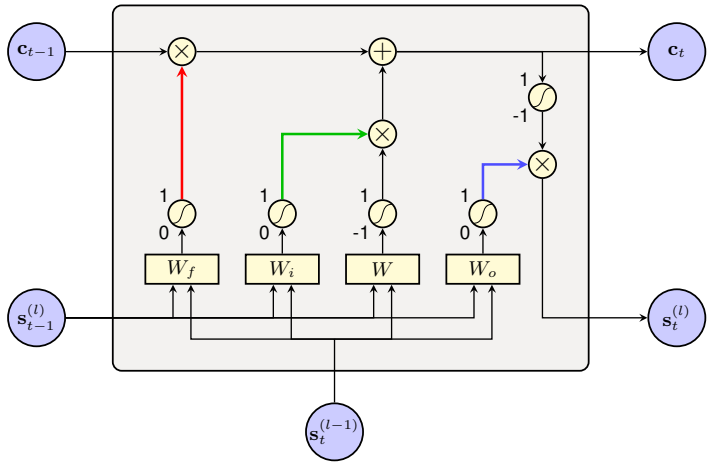


Figure 4.4. Block diagram of a single time step of an LSTM layer. Useless information is removed from the cell state input ($\mathbf{c}_{t-1}$) by multiplying it by the forget gate activation (red line). The input gate activation (green line) selects what new information is added to the cell state. The output gate activation (blue line) controls what is written to the hidden state ($\mathbf{s}_t^{(l)}$), which is also the output of the layer. The input is usually a combination of the hidden state of the previous time step ($\mathbf{s}_{t-1}^{(l)}$) and the output of the previous layer ($\mathbf{s}_t^{(l-1)}$).

useless information from the cell state to avoid the cell state growing in an unbound fashion [27].

Figure 4.4 depicts the internal structure of an LSTM layer with input, output, and forget gates. The gates use sigmoid activation, while the actual information is squashed using hyperbolic tangent. These functions are actually very similar, but sigmoid values are between 0 and 1, while hyperbolic tangent is between -1 and 1. All the gates use the same input, which depends on how the network is defined.

Typically the input to LSTM is a combination of the hidden state of the previous time step ($\mathbf{s}_{t-1}^{(l)}$) and the output of the previous layer ($\mathbf{s}_t^{(l-1)}$). In case of a language model, the latter is usually the word embeddings produced by the projection layer. Each gate needs two weight matrices for the two inputs, W and U, and a bias vector b. (Figure 4.4 shows just one weight matrix per gate.) For example, the activation of the forget gate at time step t can be written:

$$g_f(\mathbf{s}_t^{(l-1)}, \mathbf{s}_{t-1}^{(l)}) = \sigma(W_f \mathbf{s}_t^{(l-1)} + U_f \mathbf{s}_{t-1}^{(l)} + \mathbf{b}_f), \tag{4.11}$$

Cell state is multiplied elementwise by the activations of the forget gate, essentially setting selected cell state units to zero. Then information selected by the input gate is added to the cell state. A nonlinearity is applied to the cell state and it is multiplied by the output gate activations

to produce the next hidden state. However, only the forget gate and input gate affect the next cell state.[1]

LSTM was used in the recurrent neural network language models in Publications IV and V.

4.4.2 Highway Networks

LSTM facilitates information flow in recurrent layers through a sequence of time steps. Highway networks [87] use a similar idea, but try to improve information flow across layers in a (vertically) deep network. The same idea can be used in different types of layers. Consider a layer that performs the function $f(\mathbf{s}^{(l-1)})$. For example, in a feedforward network f would perform an affine transformation and then apply a nonlinear activation function. A highway network uses gates to control whether the layer actually transforms a signal using f, or passes it unchanged to the next layer. Instead of using two gates to control which value to output, the authors suggest using a single gate and its negation. Below $g_t(\mathbf{s})$ is the *transform gate*, which expresses which input signals are transformed:

$$\begin{aligned} f(\mathbf{s}^{(l-1)}) &= \tanh(W\mathbf{s}^{(l-1)} + \mathbf{b}) \\ g_t(\mathbf{s}^{(l-1)}) &= \sigma(W_t\mathbf{s}^{(l-1)} + \mathbf{b}_t) \\ \mathbf{s}^{(l)} &= g_t(\mathbf{s}^{(l-1)}) \odot f(\mathbf{s}^{(l-1)}) + (1 - g_t(\mathbf{s}^{(l-1)})) \odot \mathbf{s}^{(l-1)} \end{aligned} \tag{4.12}$$

The gate learns the parameters W_t and $\mathbf{b}_t$ that select the output between the layer's input and its activation. Essentially this means that the network is optimizing its depth. The dimensionality of the activations never change in a highway network. In Publication V, the NNLM included a four-layer-deep highway network after an LSTM layer.

4.5 Cost Functions and Softmax Approximations

A *cost* or *loss* is a function of the network parameters that the training tries to minimize. The cost function defines what kind of representation the neural network tries to learn from the data. It may also have a huge

[1]The recurrent loop of the cell state was called the constant error carousel (CEC) in the original publication. It did not include the forget gate, meaning that CEC units were only changed through addition. In such case the partial derivatives of the cell state with respect to the cell state at an earlier time step are zero. An error signal may stay in the CEC indefinitely, but a new signal may be added through the output gate.

impact on the training and inference speed in a softmax classifier. This section describes popular cost functions that have been used in neural network language models, and approximations for improving the speed of the softmax normalization.

4.5.1 Cross-Entropy Cost

The cost function that is most widely used in classification tasks is cross-entropy. In such a setting, the final layer uses the softmax activation function (Equation 4.1) to produce a valid probability distribution. In case of a language model, there are as many outputs as there are words in the vocabulary. We would like the output corresponding to the target word to be as close to one as possible, and all the other outputs to be as close to zero as possible. In other words, the desired output distribution at time step t is

$$y_i^* = \delta_{iw_t}, \tag{4.13}$$

where δ is the Kronecker delta function and w_t is the vocabulary index of the target word.

Cross entropy of two probability distributions measures how similar the distributions are. The cost function is supposed to measure how well the actual output distribution $\{y_i\}$ matches the desired distribution, and is defined as

$$\mathcal{H}(\{y^*\}, \{y\}) = -\sum_i y_i^* \log y_i = -\log y_{w_t}. \tag{4.14}$$

The summation in Equation 4.14 reduces to the negative log probability of the target word, as all the other terms will be zero. The cost of a word sequence will be the sum of the negative log probabilities of the words. Using this cost function is also equal to optimizing the cross entropy of the training data $\mathbf{w} = w_1 \dots w_t$, which is obvious when the probability of a word sequence (Equation 3.2) is substituted into the equation for empirical cross entropy (Equation 3.19):

$$\mathcal{H}(\mathbf{w}, p) = -\frac{1}{T} \log p(\mathbf{w}) = -\frac{1}{T} \sum \log p(w_t \mid w_1 \dots w_{t-1}) \tag{4.15}$$

To see how this directly relates to improving perplexity, recall from Section 3.6 that the perplexity of a word sequence is defined as the exponent of cross-entropy.

Despite its simplicity, the cross-entropy cost in Equation 4.14 is useful, because the softmax normalization in Equation 4.1 effectively causes the target preactivation a_{w_t} to contribute negatively to the cost, and all preactivations to contribute positively to the cost:

$$\mathcal{C}(\theta, w_t) = -\log y_{w_t} = -\log \frac{\exp(a_{w_t})}{\sum_j \exp(a_j)} = -a_{w_t} + \log \sum_j \exp(a_j) \quad (4.16)$$

Computing a single normalized output requires computing the exponents of all the N preactivations of the output layer, N being the size of the vocabulary.

Gradient descent moves the parameters to the direction of the negative gradient. This can be seen as a combination of positive reinforcement for the target word preactivation a_{w_t}, and negative reinforcement for all preactivations a_j. The negative reinforcements are weighted by the softmax outputs, y_j from Equation 4.1 [7]:

$$\begin{aligned} -\nabla_\theta \mathcal{C}(\theta, w_t) &= \nabla_\theta a_{w_t} - \frac{1}{\sum_j \exp(a_j)} \sum_j \exp(a_j) \nabla_\theta a_j \\ &= \nabla_\theta a_{w_t} - \sum_j y_j \nabla_\theta a_j \end{aligned} \quad (4.17)$$

All N preactivations have to be backpropagated for computing the gradients.

4.5.2 Importance Sampling

Complexity of the gradient computation at the output layer is proportional to the size of the vocabulary, which can be large. In shallow networks this can be a significant proportion of the total training cost. One would think that a good enough approximation could be obtained without computing the network output for *every* word. This is exactly the idea behind several sampling-based training methods. They are particularly attractive with large vocabularies, since their computational cost depends on the number of sampled words, not on the vocabulary size. They are usually only used to speed up training, and full softmax is taken during inference.

The summation in Equation 4.17 can be seen as the expectation of the gradient of the preactivations $\{a_w\}$ under the probability distribution of the model $p(w) = y_w$. Importance sampling [7] is a Monte Carlo method that approximates that expectation by taking it over a random subset of the vocabulary. The difficulty with this method is that in order for the

approximation to be as accurate as possible, the words for the summation should be sampled from a distribution that is similar to the output distribution of the model. An estimator is also needed for y_w of the sampled words that does not require computing all the outputs.

4.5.3 Noise-Contrastive Estimation

Noise-contrastive estimation (NCE) [33] turns the problem from separating the correct class from the other classes into the binary classification problem of separating the correct class from noise. This quite popular method can greatly improve training speed on shallow NNLMs, as the computational cost of training them is dominated by the output layer [14].

For each word w_t in training data, a *noise* word w_t^n is sampled from a known distribution $p^n(w)$, e.g. the uniform distribution. When considering the union of training and noise words, the combined distribution is $p^c(w) = \frac{1}{2}\, p^d(w) + \frac{1}{2}\, p^n(w)$. The data distribution $p^d(w)$ also depends on the context $w_0 \dots w_{t-1}$, as may the noise distribution $p^n(w)$, but for brevity the distributions are not indexed with the time step t in the following equations. The true data distribution is unknown, so the output of the model is used instead:

$$p^d(w) = y_w = \frac{\exp(a_w)}{\sum_j \exp(a_j)} = \frac{\exp(a_w)}{Z_t} \tag{4.18}$$

Because we want to avoid computation of the sum in the denominator, NCE treats the normalization term Z_t as a model parameter that is learned along with the network weights.[2] As it seems, this approach is not as easy to implement in RNNs, since Z_t depends on the context. However, it turns out that fixing $Z_t = 1$ does not harm the performance of the resulting model, because the network automatically learns to adjust to the constraint [65].

Let us consider the probability of a sample that comes from the combined distribution $p^c(w)$ being a training or noise word. This is indicated by the auxiliary label $C = 1$ for training words and $C = 0$ for noise words.

[2] It is not possible to treat the normalization constant as a network parameter in normal softmax output, because the cost in Equation 4.16 can be made arbitrarily low by making the denominator go toward zero.

$$
\begin{aligned}
p^c(C=1 \mid w) &= \frac{p^c(C=1)\, p^c(w \mid C=1)}{p^c(w)} \\
&= \frac{p^d(w)}{p^d(w)+p^n(w)}
\end{aligned}
\tag{4.19}
$$

$$
p^c(C=0 \mid w) = 1 - p^c(C=1 \mid w) \tag{4.20}
$$

These probabilities can be written using the sigmoid function $\sigma(x) = 1/(1+\exp(-x))$:

$$
\begin{aligned}
p^c(C=1 \mid w) &= \frac{p^d(w)}{p^d(w)+p^n(w)} \\
&= \frac{1}{1+p^n(w)/\,p^d(w)} \\
&= \frac{1}{1+\exp(\log p^n(w) - \log p^d(w))} \\
&= \sigma(G(w))
\end{aligned}
\tag{4.21}
$$

$$
p^c(C=0 \mid w) = 1 - \sigma(G(w)), \tag{4.22}
$$

where $G(w) = \log p^d(w) - \log p^n(w)$.

For a binary classification problem, the cross-entropy cost function, defined over all the training words $\mathbf{w}$ and noise words $\mathbf{w^n}$ is:

$$
\begin{aligned}
\mathcal{C}(\theta, \mathbf{w}, \mathbf{w^n}) = -\sum_{w \in \{\mathbf{w}, \mathbf{w^n}\}} & \big[C_w \log p^c(C=1 \mid w) \\
& + (1-C_w) \log p^c(C=0 \mid w)\big]
\end{aligned}
\tag{4.23}
$$

This results in the following cost for a single pair of training word w_t and noise word w_t^n:

$$
\mathcal{C}(\theta, w_t, w_t^n) = -\log \sigma(G(w_t)) - \log(1-\sigma(G(w_t^n))) \tag{4.24}
$$

The cost function can be simplified using the softplus function $\zeta(x) = \log(1+\exp(x))$:

$$\begin{aligned}\log\sigma(x) &= \log\frac{1}{1+\exp(-x)} \\ &= -\log(1+\exp(-x)) \\ &= -\zeta(-x)\end{aligned} \tag{4.25}$$

$$\begin{aligned}\log(1-\sigma(x)) &= \log(1-\frac{1}{1+\exp(-x)}) \\ &= \log(1-\frac{\exp(x)}{\exp(x)+1}) \\ &= \log(\frac{\exp(x)+1-\exp(x)}{\exp(x)+1}) \\ &= \log(1)-\log(\exp(x)+1) \\ &= -\zeta(x)\end{aligned} \tag{4.26}$$

$$\mathcal{C}(\theta, w_t, w_t^n) = \zeta(-G(w_t)) + \zeta(G(w_t^n)) \tag{4.27}$$

4.5.4 Generalization to Larger Noise Sample

The NCE objective function can be made more accurate by sampling $k > 1$ noise words per training word [34]. The combined distribution of training and noise words is now

$$p^c(w) = \frac{1}{k+1}\,p^d(w) + \frac{k}{k+1}\,p^n(w), \tag{4.28}$$

and the posterior probabilities of the data and noise classes become

$$\begin{aligned}p^c(C=1 \mid w) &= \frac{p^c(C=1)\,p^c(w \mid C=1)}{p^c(w)} \\ &= \frac{p^d(w)}{p^d(w)+k\,p^n(w)} \\ &= \frac{1}{1+k\,p^n(w)/\,p^d(w)} \\ &= \sigma(G(w))\end{aligned} \tag{4.29}$$

$$\begin{aligned}p^c(C=0 \mid w) &= 1-p^c(C=1 \mid w) \\ &= \frac{k\,p^n(w)}{p^d(w)+k\,p^n(w)} \\ &= \frac{1}{p^d(w)/\,(k\,p^n(w))+1} \\ &= \sigma(-G(w)),\end{aligned} \tag{4.30}$$

where $G(w)$ has been redefined $G(w) = \log p^d(w) - \log k\,p^n(w)$.

Compared to Equation 4.16, evaluation of a_j for all the vocabulary words is reduced to $k+1$ evaluations per training word. In Publication V, the best

Finnish word model was trained using NCE, but generally NCE training seemed unstable.

4.5.5 BlackOut

Another method that was implemented in Publication V, BlackOut [44], is similar to NCE but simpler to define. It replaces softmax with a weighted approximation that normalizes the target word only on $k+1$ words, the target word and the noise words:

$$\begin{aligned} q_w &= \frac{1}{p^n(w)} \\ \tilde{p}(w) &= \frac{q_w \exp(a_w)}{q_w \exp(a_w) + \sum_j q_j \exp(a_j)}, \end{aligned} \tag{4.31}$$

where the summation is over the sampled noise words.

The cost discriminates explicitly the target word from the noise words:

$$\mathcal{C}(\theta, w_t, \mathbf{w}_t^n) = -\log \tilde{p}(w_t) - \sum_{w' \in \mathbf{w}_t^n} \log(1 - \tilde{p}(w')) \tag{4.32}$$

In our experiments BlackOut was slightly faster, but less stable than NCE, and we could not get results from this method.

4.5.6 Unnormalized Models

The softmax approximations described in the previous sections speed up training of neural networks. In some applications inference speed is more important than training time. For example, there are situations where transcription of television broadcasts should be done in real time. It is only a matter of time until NNLMs find their way into real-time applications.

The model does not have to be correctly normalized in order to be useful for speech recognition, as long as it ranks the good sentences above those that are spoken less likely. Variance regularization uses a training objective that penalizes models based on how far they are from the normalized model before the normalization [83]. It learns a model that is approximately normalized, so that during inference the normalization is not needed. Another approach is to predict the normalization term (Z_t in Equation 4.18) from the network inputs along with the unnormalized word probabilities [81].

4.5.7 Hierarchical Softmax

A different approach for improving output layer performance is to factor the softmax function into a hierarchy of two or more levels. Originally the idea emerged from maximum entropy language modeling [30]. Maximum entropy models, like softmax, have a log-linear form that suffers from the computational cost of explicit normalization (see Section 3.2.2). A hierarchical decomposition of softmax improves both training and inference speed of NNLMs [67].

Each level of the hierarchy takes softmax over classes of words, narrowing down the set of words that one class encompasses. If a single softmax layer had N outputs, in a two-level hierarchy each softmax function would have only $\sqrt{N}$ outputs. The first level would compute probabilities over $\sqrt{N}$ classes, and the second level over $\sqrt{N}$ words in the classes.

Usually we are interested only in the probability of a particular word. Then the computation is reduced dramatically, because we need to compute only one softmax on each level. Assuming the two-level hierarchy, the first level computes the probability of the target class and the second level computes the probability of the target word inside the class. The actual output probability is the product of these two probabilities. The neural network inputs and outputs are words, which makes it more accurate than class-based models that use unigram estimates for probabilities inside a class.

Hierarchical softmax was used in Publication V for vocabularies as large as 500,000 words. Training was stable and the model performance seemed to be close to normal softmax.

4.6 Combining Data Sources

When building a practical speech recognition system, the amount of training data usually limits the accuracy that can be achieved. Language modeling data can be collected from many sources, for example newspaper archives and the Internet. This was the case with conversational Finnish as well. The models were trained on data from many different sources of varying size and quality. There is a well-established method for combining n-gram models estimated from different corpora, as explained in Section 3.5. Because neural networks use a complex nonlinear function to predict the probabilities, such a method for merging multiple neural networks into

one does not exist.

Certainly multiple neural networks can be trained on different data sets and the probabilities can be combined by interpolation. The best published results on speech recognition tasks are often reached by combining many different models. However, training different models from many data sets would be laborious, since the hyperparameters need to be optimized for each data set separately, and such a system would be impractical to use, requiring evaluation of every component model.

Often a large amount of general data can be collected from sources that do not match the task so well, for example the Internet or newspaper archives. The amount of data that matches the task well, for example transcribed data is typically a lot smaller. If all training data is given equal importance, the large amount of worse data will dominate the model. The problem is analogous to model adaptation, where a model is trained on a large set of general data and later adapted using data specific to the problem domain or adaptation data recorded from the person using the system.

Publication III summarizes the methods that have been used for combining different data sets in neural network training:

- On each training epoch, use only a subset of the worse-quality data set that is randomly sampled before the epoch.

- Train first on general data and toward the end of the training on in-domain data.

- Train a model on general data and adapt it for example by adding a layer that is trained on in-domain data, while keeping the rest of the model fixed.

- Train a multi-domain model by adding a small set of parameters that are switched according to the domain. This requires that the domain is known during inference.

In Publication III the multi-domain approach and adding an adaptation layer were explored for training language models in the absence of large in-domain corpora. An additional input was added to the network that identifies the data set. This input was mapped to a domain-specific vector

that modified the hidden layer preactivations.

Publication V evaluated two methods for combining conversational Finnish data sets. A novel method was introduced that weights the parameter updates based on the data set, according to predefined weights. In other words, the learning rate was dependent on from which data set the mini-batch was taken from. This method was compared to randomly sampling a subset of the large data set in the beginning of each training epoch [79, p. 206]. Both approaches suffer from the fact that there are no methods for optimizing the weights and sampling coefficients. An advantage of random sampling is that it also makes training faster.

4.7 Implementation of TheanoLM

TheanoLM is a neural network language modeling toolkit implemented using Theano [1], a Python library for evaluating mathematical expressions. It was used in Publications IV and V and released to the public.

Theano provides an interface that is similar to the well-known scientific computing package NumPy, for expressing functions such as the mapping of neural network inputs to outputs. However, while NumPy performs mathematical operation on numerical data, Theano functions are defined for symbolic matrices and higher-dimensional tensors. They are stored as a graph of the computation required to produce the output from the input, which is made possible by the high level of abstraction in Python. By creating a computation graph instead of performing the actual computation directly, it is possible to perform differentiation using the backpropagation algorithm.

For example, the Python code in Listing 4.1 shows how a highway network (see Section 4.4.2) layer function can be defined using Theano interface. It defines the preactivations as an affine transformation of the layer input and parameters, all of which are symbolic variables. The preactivations `s` and `t` of the normal output and the transform gate are computed together, because large matrix operations can be parallelized efficiently. The weight and bias are concatenations of the individual weight matrices and bias vectors. Then the computed preactivation matrix is sliced into two parts. `s` uses hyperbolic tangent activation and `t` uses sigmoid activation. Finally the output is defined to be either the normal output or the input of the layer, as selected by the sigmoid gate.

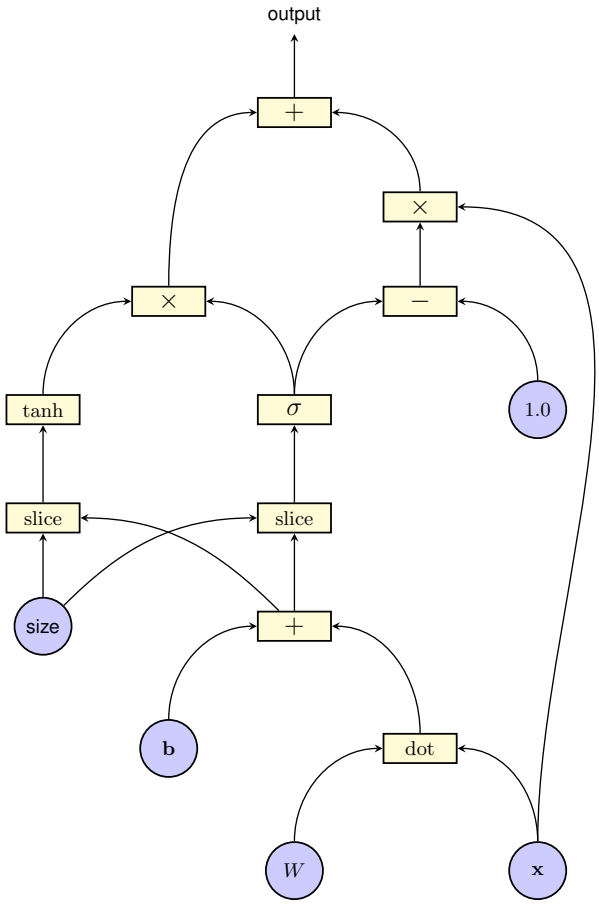


Figure 4.5. Computation graph for a highway network layer before optimizations.

Listing 4.1. Theano implementation of a highway network layer

```
preact = tensor.dot(layer_input, weight) + bias
s = tensor.tanh(preact[:, :size])
t = tensor.nnet.sigmoid(preact[:, size:])
layer_output = s * t + layer_input * (1.0 - t)
```

The code in Listing 4.1 would produce a computation graph that is depicted in Figure 4.5. After creating the graph, Theano applies various optimizations that make it more compact.

After the parameters are selected, the output of the network can be computed by feeding the graph with actual inputs and performing a forward pass. Section 4.3.2 describes how the backpropagation algorithm can be used to compute the gradient of a nested function at given input. Theano takes this approach a step further. Instead of computing the gradient using given input values, it creates a computation graph that describes the

gradient given a symbolic input tensor. The advantage is that the gradient can be further differentiated to obtain higher order derivatives.

Theano automatically performs the computations on a GPU, when available. GPUs can perform operations on large matrices efficiently in parallel. It is useful to process as much data in parallel as is possible to fit in the GPU memory. TheanoLM cuts sentences according to the maximum sequence length parameter, which limits the distance to which backpropagation is truncated. Several sequences are packed in a mini-batch to make computation more efficient, controlled by the batch size parameter.

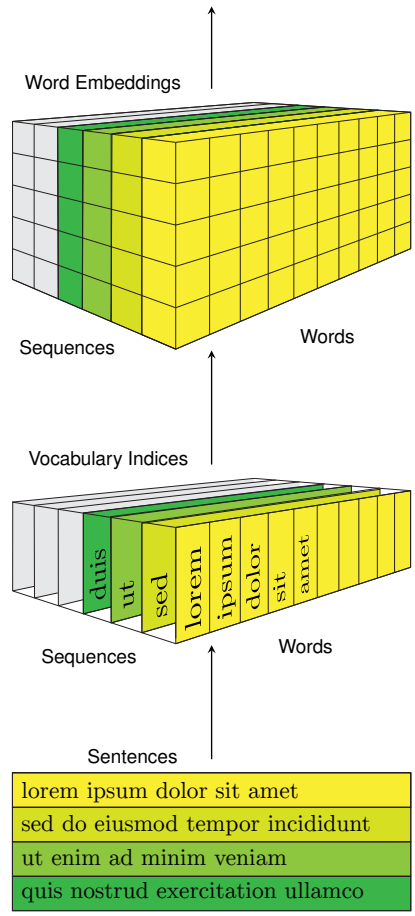


Figure 4.6. TheanoLM processes data in mini-batches.

Every layer in a TheanoLM network processes one mini-batch at a time, as shown in Figure 4.6. The input of the network is a matrix of vocabulary indices, whose first dimension is the time step, and second dimension is the sequence index. When a mini-batch contains sequences of different length, the shorter sequences are padded and a binary matrix is used to mask out elements that are past the sentence end. The projection layer transforms the input matrix to a three-dimensional tensor, by mapping the vocabulary indices to continuous-valued vectors.

Each successive layer defines an operation that takes one or more three-dimensional tensors as input and produces a three-dimensional tensor of activations. The input of a layer is defined by the network topology, and can include outputs of any previously defined layers. The layer function can be freely defined using input from every time step, within the limits of the mini-batch. Recurrent layers typically define the output as a recursion that uses only past context, but bidirectional and convolutional layers may use inputs from both directions.

As discussed in Section 4.4, the subgradients of the training cost may

be very small or large and this can cause numerical instability. A simple step taken by TheanoLM to avoid the gradients exploding is to normalize the subgradient to a specified maximum norm if the norm gets larger [73, p. 1315]. It is important to perform the normalization after any adaptive gradient method is applied, because adaptive gradient methods may produce very large values when the subgradients have been small on average. An action that is taken to avoid numerical instability caused by very small gradients is to add a small constant to the norm of gradients before dividing by that value. Similarly a small value is added to probabilities before taking the logarithm.

4.8 Using NNLMs to Rescore Decoder Output

Evaluation of neural network language models is currently still too slow to be used in the first decoding pass. In practice they are applied in a second pass that is performed on a list of n best hypotheses (n-best list) or a word lattice. In Publication IV we rescored n-best lists, except with RWTHLM, which was able to rescore word lattices. In Publication V a word lattice decoder was developed in TheanoLM.

The challenge in rescoring word lattices using RNN language models is that in principle RNNs use all the context from the beginning of the sentence up to (but not including) the word to be predicted. If the entire history is considered, all paths through the lattice are unique, and the search space is as large as an n-best list that contains all the possible paths through the lattice. Thus some heuristics are needed to keep the search space reasonable.

The decoder is implemented after the conceptual model of token passing. In the beginning an initial token is created in the start node. Then the nodes are processed in topological order. Three types of pruning are applied to the tokens of a node, before propagating the tokens to the outgoing links [90]:

- **N-gram recombination**. The contexts of two tokens in the same node are considered similar if the n previous words match, and only the best token is kept. n is selected experimentally.

- **Cardinality pruning**. Only a specific amount of best tokens are kept in each node.

- **Beam pruning**. Tokens whose probability is low compared to the best token are pruned, similar to beam pruning in speech recognizers.

N-best lists and word lattices contain the original n-gram model probabilities from the first decoding pass. Usually the best result is obtained by interpolating the NNLM probabilities with the original probabilities. A straightforward approach is linear interpolation of log probabilities:

$$\log p^*(\mathbf{w}) = (1-\lambda)\log p_{bo}(\mathbf{w}) + \lambda \log p_{nn}(\mathbf{w}) \tag{4.33}$$

The resulting probability is the product of the component probabilities raised to the power of the weights:

$$p^*(\mathbf{w}) = p_{bo}(\mathbf{w})^{1-\lambda} * p_{nn}(\mathbf{w})^{\lambda} \tag{4.34}$$

However, Equation 4.34 is not a probability distribution. In order to get real probabilities, it should be normalized so that the probabilities sum to one:

$$\log p(\mathbf{w}) = \log p^*(\mathbf{w}) - \log \sum_{\mathbf{w}} p^*(\mathbf{w}) \tag{4.35}$$

This kind of interpolation is called log-linear interpolation [47]. Computing the normalization can be challenging when w is a word sequence. We used log-linear interpolation in Publications IV and V to combine NNLM probabilities with lattice probabilities. However, as correct normalization is not important in speech recognition, we used Equation 4.33 without normalization.

Linear interpolation is mathematically simple, but when implementing it, care needs to be taken when converting the log probabilities into probabilities, in order to avoid floating point underflow.

$$p(\mathbf{w}) = (1-\lambda)\, p_{bo}(\mathbf{w}) + \lambda\, p_{nn}(\mathbf{w}) \tag{4.36}$$

4.9 Details on the NNLMs Used in This Thesis

The shortlist approach was used in the NNLM experiments in Publication III. A feedforward neural network predicted the probability for the 1024 most frequent words (or subwords). The input vocabulary was larger and included a special token for out-of-vocabulary words. The input of the network was three previous words and the data was processed in mini-batches

of 200 samples. The network was trained only on n-grams that end in an in-shortlist word, so the output vocabulary did not include an out-of-shortlist (OOS) token. The probability for OOS words was predicted by a 4-gram back-off model alone. In this approach, the in-shortlist word probabilities given by the NNLM have to be normalized using the probabilities given by the n-gram model [79, p. 203].

In Publication IV, recurrent neural networks trained using three different toolkits were compared. In all cases, 2000 word classes were created from a vocabulary of 2.4 million word forms to keep the computation feasible. However, RNNLM used a different frequency-based method for deriving the classes and a hierarchical softmax output layer [62].

In Publication V, the shortlist approach was used as a baseline for RNN language models. Words, subwords, word classes, and subword classes were used as the language modeling unit. The sequence length in a mini-batch was limited to 25 units for efficiency. The input and output vocabularies were identical, both using a special OOS token.

The OOS token models the total probability mass of all OOS words. The correct way to compute the probability of any OOS word is to divide the probability predicted for the OOS token by the NNLM according to the probabilities predicted for the OOS words by the n-gram model. It requires evaluation of the n-gram probabilities of all the OOS words in the same context. To keep the implementation simple and fast, unigram probabilities were used for OOS words. In the article we tried also a simple approximation of replacing the OOS probability with a 4-gram probability [72], but it did not work well because the probability mass that the two models predicted for the OOS words was too different.

In Publication III the feedforward network was trained using SGD. In Publications IV the other toolkits used SGD, but with TheanoLM we used Adagrad. Adagrad was found to converge fast in many cases and was also used in Publication V.

Sampling-based approximations of softmax were evaluated for Publication V. They were less stable in our experiments and the speed benefit was smaller than expected. On reason for the disappointing speed is that those methods cannot be implemented on a GPU as efficiently using dense matrix products. The network contained an LSTM layer and a four layers deep highway network. The deeper architecture also means that relatively less can be gained by improving the speed of the output layer.

One reason for the instability may be the distribution of words in the

data. The sampling distribution and the sample size have a great impact on the speed of convergence and numerical stability. Sampling from a uniform distribution failed to converge to a good solution. The unigram power distribution [63, p. 3114] produced clearly better models. However, when using the power distribution, the training time grew because the multinomial sampling implementation in Theano was slow.

Nevertheless, the best Finnish shortlist model was trained using NCE. We were unable to get any results using BlackOut.

5. Collecting Conversational Finnish Data

5.1 Aalto University DSPCON Corpus

The work on this thesis started by collecting conversational Finnish training and test data. The data collection continued until 2016. Students of the Digital Signal Processing course at Aalto University were asked to have conversations in pairs. Each student recorded at least 20 utterances, and transcribed their own utterances. Initially the quality of the transcripts was quite varying, so they were reviewed by researchers and errors were corrected. Errors were found by trying forced alignment of the transcript on the audio with a small beam.

Part of the data was dedicated as development and evaluation sets. The intention was that the evaluation set could be used to track the progress in conversational Finnish ASR during and after this thesis work. For this purpose, the task was intentionally made more challenging and generic by transcribing some very informal radio conversations and adding these to the development and evaluation sets. Some of them contain also music in the background. Alternative word forms for scoring purposes were added to the development and evaluation set transcripts by the author.

The data set is not very large, in total 9.8 hours of audio, but perhaps even more important to speech recognition is that the number of different speakers is quite high. 5281 utterances were spoken by 218 different male students and 24 female students. The audio was recorded using headsets, eliminating most of the background noise, but some noise is still audible. The corpus has been made available for research purposes.[1]

[1]DSPCON corpus metadata can be accessed on META-SHARE:
`http://urn.fi/urn:nbn:fi:lb-2015101901`
The corpus can be downloaded from the Language Bank of Finland:
`http://urn.fi/urn:nbn:fi:lb-201609191`

5.2 Collecting Language Modeling Data from the Internet

Hours of transcribed speech is required for training phoneme models, and even that is usually not enough for estimating good language models. The fact that the acoustics and language are modeled separately (see Section 2.2) greatly simplifies data collection, as the texts used for estimating language models do not need to be transcriptions of recordings that we possess; it is enough that the text is similar to how we speak. Written conversations generally are somewhat different to transcriptions—written text is usually better structured, words are often less colloquial, and speech disfluencies such as “uh” and “um” are omitted—but so much data is available on the Internet that with clever data selection it can provide valuable resources for modeling conversational language.

The most important data set for conversational Finnish language modeling used in this thesis consisted of 2.7 billion words or 280 million sentences after normalization. In Publication III, the data was filtered into a set of 76 million words or 9.0 million sentences. The filtered data was combined with DSPCON corpus to form the language modeling data in Publications IV and V.

First part of the data was collected using Google searches similar to Bulyko et al. [12], but this method was soon found out to be too inefficient for finding large amounts of conversational data. The rest of the data was obtained by crawling conversation sites using customized Python scripts. The scripts were written on top of the Scrapy application framework, which is designed for extracting structured data. The scripts, called web spiders, start from a top-level web page and follow the links to conversation areas and individual conversation threads.

Crawling static web pages is straightforward. A spider implementation is required only to specify the rules for following links to different conversations, and a function for parsing messages from conversation threads. Iterating through nested elements in an HTML document is facilitated by *XPath* (XML Path Language) selectors, a mechanism for identifying elements in an HTML document or inside another element. Crawling sites that generate pages dynamically on client side is a bit more involved. Selenium WebDriver library was used to construct dynamic HTML pages by controlling a web browser that supports JavaScript.

The spiders extracted identifiers from the HTML code, uniquely identifying each message and conversation. The purpose was to avoid saving the

same message multiple times, and enable filtering to be performed per message or per conversation, instead of per sentence. Estimating probabilities from individual sentences would be unreliable. Publication II concluded that filtering per message is better than filtering per conversation.

For Publication III, a data set of Estonian conversations was collected using the same method. It consisted of 340 million words or 33 million sentences after normalization. By choosing the filtering method that improved speech recognition accuracy the most, data size was reduced to 82 million words or 6.6 million sentences.

5.3 Text Normalization

Text collected from the Internet is very heterogeneous. Normalization is needed to get suitable training data for the task in hand. Preparing the normalization pipeline can take a lot of work, but is less interesting from scientific perspective, so it is often left aside from publications. The purpose of the preprocessing is to format the text as closely as possible to how a person would pronounce it. A complex set of regular expressions were used to process the text. Below is a list of the main steps used to process the Finnish web data:

- Some sanity checks were performed to filter out garbage. A too long line, a too long word, or a line that repeats the same sequence more than three times, was taken as indication that the line is not proper text.

- Headers and annotations used by some conversation sites to mark e.g. text style were filtered out, as well as emoticons and program code.

- Abbreviations, contractions, and acronyms were expanded to their original form. Numbers, number ranges, dates, unit symbols, etc. were expanded to how they are pronounced. Inflected acronyms and numbers, denoted by a subsequent colon and suffix, were expanded to the correctly inflected word forms.

- In some cases, where punctuation marks would be pronounced—for example in real numbers and domain names—they were expanded to letters. In other cases, periods, colons, semicolons, and parentheses were taken as sentence breaks, and other punctuation marks were removed.

- Words that contain a sequence of letters that does not exist in Finnish language (including loan words) were either deleted or corrected. Repetition of a single character (e.g. “noooo”) was shortened, but in other cases the word was deleted.

- All letters were translated to lower case and all characters except letters were removed. As online conversations are often incorrectly capitalized, this data was not used for learning correct capitalization, and it was not expected in the test data.

How to expand numbers to letters is a surprisingly difficult question. Section 2.5.2 lists 20 ways in which the number *19* can be pronounced. Ideally we would like the occurrence of *19* in the training text to increase the probability estimate of all the pronunciation variants in that context. Properly solving the problem would require significant effort. Our simple way to sidestep the problem was to replace all numbers with the standard Finnish pronunciation. The effect on the overall error rate is not large, as it is quite likely that numbers are recognized correctly even if the speaker uses a slightly different pronunciation.

5.4 Text Filtering

There are many conversation sites on the Internet, with huge amounts of text. Even when downloading conversations only from a specific site that should in principle match the targeted language well, free-form conversations always contain different writing styles and often even different languages.

The task of selecting text that is similar to some in-domain example text, from a larger corpus that is not domain specific, has been studied before in different contexts. In this case the in-domain text is transcribed conversations. It is used to select text segments from a large data set of written conversations from the Internet. The text segments can be sentences, or for example messages in a social media site.

The question of how to select segments that are similar to the in-domain text is not trivial to answer. Intuitively the selected segments should contain many of the same words as the in-domain data. An approach that is common in information retrieval is to compare tf–idf statistics of the text segments (documents) and the query, for example using cosine similarity

[56]. This is a heuristic technique, but popular because of its simplicity.

It is possible to derive approaches that are theoretically better justified, by regarding words as samples from some random phenomenon, which is not know, but can be approximated by a language model. Information theory provides measures for comparing probabilistic models and samples drawn from a probability distribution. Several different ways to use these measures for the text selection task have been proposed in the past. The most straightforward technique involves estimating a language model from the in-domain data and computing its empirical perplexity or cross-entropy on the text segments [54]. (Recall from Equation 3.20 that perplexity is simply the exponent of cross-entropy, so they are equivalent with regard to text selection.)

Perplexity measures how well a model predicts a text segment. When the model is estimated on the in-domain data, perplexity of a text segment is related to how similar the in-domain data and the text segment are, but does not explicitly measure the objective of the task. The aim is to find a set of text segments, so that a model estimated on them gives a low perplexity for the in-domain data. Klakow proposed a method that trains a language model from the unfiltered corpus, and from the same data with one text segment removed at a time [48]. Each model is used to compute the perplexity of the in-domain data. If the perplexity of the in-domain data is higher when a segment is removed, the segment should be included in the selection set.

Another way to see the problem is as a classification task with only positive examples (text segments that are in-domain data) and unlabeled examples (text segments of which we do not know whether they are in-domain data or not). The probability of a random text segment being in-domain data can be expressed using a model of the in-domain data and a model of the unlabeled data, by applying the Bayes rule. This leads to the selection criterion proposed by Moore and Lewis [66] that compares the perplexity of an in-domain model to the perplexity of a model estimated from the same amount of unfiltered web data.

All the above methods have one shortcoming: They compute a score for every text segment individually, and then select all the segments whose score is above some fixed threshold. Text segments that are very probable according to the in-domain model are selected, regardless of how many similar sentences have already been selected. Text segments that would rarely occur according to the in-domain model are never selected. In other

words, the distribution of sentences in the selection set is biased toward the high-probability ones.

Sethy et al. abandoned the score-and-filter approach [82]. They aimed to build a selection set whose distribution of sentences is similar to the distribution in the in-domain data. The similarity of two distributions can be measured using relative entropy. Relative entropy of two language models is defined in Equation 3.21. In this case, a unigram model is assumed, and only the change in relative entropy caused by adding the new words is computed, making the decision rule fast to use. As noted in Publication III, the algorithm itself cannot be parallelized, but multiple passes can be run in parallel.

Publications II and III compare different methods for filtering language model training data that has been downloaded from the Internet. The algorithms were slightly modified to work with very large data sets and agglutinative languages.[2] Subword models were used, instead of word models, as a solution to estimating the probability of unknown words (see Section 3.6). This was necessary especially in methods that train models from the development data, because our development set was very small. An efficient implementation of Klakow's algorithm was presented that requires counting once all the words that occur in the training data, and then for computing the score of a text segment, counting the occurrences of the development set words in the text segment.

In the Finnish task, the Sethy's algorithm produced the smallest data set and best WER. In the Estonian task, Klakow's algorithm gave the best WER, perhaps because there were more development data available.

[2]The implemented methods have been made available:
`https://github.com/senarvi/senarvi-speech/tree/master/filter-text`

6. Conclusions

In 2009, a few years before I started working on this thesis, WER close to 10 % was obtained on clean speech from Speecon corpus, and close to 20 % on a slightly more difficult SpeechDat task [75]. These can be considered good results. Attempt to recognize conversational Finnish using similar standard Finnish models in Publication II showed that there is a large mismatch in the data, WER being above 70 %.

The aim of this thesis was to develop a reasonably good recognizer for conversational Finnish. A concrete objective was set when we tried conversational English speech recognition in Publication I. A large corpus of 72 hours of meeting speech was used, and the word error rate was approximately 40 %. A realistic target was to develop a conversational Finnish speech recognizer with as good performance.

It was clear that better training data was needed. The work started by collecting a small corpus of transcribed conversations. The DSPCON corpus has been updated until 2016 and released for researchers. Development and evaluation sets were created so that the progress in conversational Finnish ASR can be tracked throughout the thesis and by researchers working on this task afterwards. Data that is very difficult for automatic speech recognizers was purposefully selected to the evaluation set.

Language model training data can be found from the Internet, but there are differences between the text found online and the language used while speaking. Data was scraped from Internet conversation sites, to find out how much Internet conversations can benefit conversational Finnish language modeling. Publication II described the problem in conversational Finnish vocabulary and the data collection efforts, and proposed solutions for selecting suitable data. First attempts to recognize conversational Finnish using the new data were made, and 57.5 % WER was obtained using only web data for language modeling, and 55.6 % by combining all

the available data.

The work was continued in Publication III. Already 2.6 billion tokens of text were collected from the Internet, so efficient methods were needed for reducing the data size by discarding data that is not relevant for the task. The size of the data set was reduced to 76 million words, yet the model performance was improving. With only web data used for language model training, 54.1 % WER was obtained. Clearly Internet conversations improved the language models, but a huge amount of data was needed for a relatively small improvement. Some idea of the relative importance of the web data and the transcribe data is given in Publication V, which reports that the EM optimization gave approximately equal mixture weights for both data sets, while there was only 61,000 words of transcribed text.

Pronunciation modeling was studied in Publication I. A method for automatically pruning pronunciations was evaluated on an English dictionary, but did not give significant improvement. The same method was used when adapting models for foreign names in Publication V. However, in conversational Finnish language models, different written forms of the same word were kept as different units.

As subword models at first did not seem to improve over word models, we studied whether class-based models could be a solution to the data sparseness. In Publication IV, interpolating a word model with a class model provided a small improvement. In Publication V, class models consistently outperformed word models even without interpolation. In the former publication the vocabulary was limited to 200,000 words, while in the latter publication the vocabulary contained more than two million words. This is important in the future when the state of the art is pushed forward by using larger models, larger vocabularies, and more computational power.

Recently deep neural networks have benefited especially conversational speech recognition because of their ability to generalize to unseen data. Existing neural network language modeling toolkits were found difficult to extend with new methods, or slow because of lacking GPU support. Publication IV presented an easily extensible language modeling toolkit, which was made possible by the very high level of abstraction in Theano library. It can also utilize GPUs to efficiently train recurrent neural networks. 48.4 % WER was achieved in the conversational Finnish task when interpolating neural network probabilities with those from a traditional n-gram model—however, the same evaluation data was used for optimizing the language model weight. In the light of the experiments, GPU training

was seen important in order to keep the training times reasonable with complex models and large data sets. GPUs also bring new difficulties, because the GPU boards offer less memory, memory transfers are slow, and care has to be taken to utilize the parallel processing optimally.

In Publication V deep neural networks were used also for acoustic modeling. Different approaches were compared that address problems with large vocabularies, especially with neural network language models. A novel result of that publication is that while the subword n-gram models did not outperform word models, neural network subword models did, even when the baseline model used a very large vocabulary consisting of millions of words. This is probably due to the ability of recurrent neural networks to model long contexts more efficiently than variable-order n-gram models. In this final paper a WER of 27.1 % was achieved in the conversational Finnish speech recognition task.

In the last paper we managed to get a WER that was under the 40 % target. To large part credit of reaching the milestone is due to the data collection and improved language models that were developed in this thesis. Meanwhile, there has been ongoing research on acoustic modeling that has benefited the whole speech recognition community, as the latest methods are implemented in the Kaldi speech recognition toolkit. These advances led us to exceed the target performance even more than was expected. Seeing how important sharing the implementations is for speech recognition research, all the code developed for the experiments in this thesis has been made publicly available. A particularly significant contribution to the community is publishing the TheanoLM toolkit that will make it easier to experiment on neural network language models.

6.1 Future Work

The accuracy that we obtained for conversational Finnish speech recognition is already useful in some applications, but many challenges were still left to be solved by future research. There is currently a huge amount of research aimed toward improving modeling with artificial neural networks. Only a few of all the interesting approaches that could be applied to language modeling were explored in this thesis. Below are some directions into which this research could be continued.

One of the most interesting recent developments in machine translation is *attention*, a mechanism that learns to select which input words are the

most important in predicting an output word [3]. It could be a solution for utilizing long-distance cues in language models too [59].

The state-of-the-art NNLMs are quite shallow networks (but recurrent), while the best computer vision models contain many layers. One interesting question is why deep networks do not seem to benefit language modeling as much. We have already started research at Aalto University on exploiting deeper networks in language modeling.

Language models traditionally predict the probability of a word given the words that precede it, obeying the chain rule in Equation 3.1. However, when rescoring n-best lists, the future words are known too. This would open up new possibilities, such as bidirectional recurrent networks [78], convolutional networks [19], and attention over both past and future words.

Lately there has been a lot of interest in generative adversarial networks in computer vision [29]. The general idea—that a neural network could learn by competing with another neural network—has also been used to teach artificial intelligence to play Go [86]. Since language models can be used to generate text, a similar technique could be used to learn better language models [99].

In practice the usefulness and performance of neural network models is limited by the computational cost of training and using them. In this thesis NNLMs were used only for rescoring the output of a speech recognizer, meaning that the quality of the result may be limited by the quality of the generates word lattices, and the process is too slow for online recognition. Future applications demand especially faster evaluation of the NNLM probabilities. One way to speed up the inference would be to use unnormalized models [83]. Both memory consumption and speed could also be improved by parallelizing the model to multiple GPUs.

Finally, conversations differ from planned speech in many other ways besides the vocabulary, and these changes are not necessarily visible in written conversations. The flow of speech in a conversation may be disfluent and usually does not follow the sentence structure of written text. Modeling these differences explicitly may be necessary to have as accurate system as possible [41].

References

[1] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *Computing Research Repository*, abs/1605.02688, May 2016.

[2] Ebru Arisoy, Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Deep neural network language models. In *Proceedings of the Workshop on the Future of Language Modeling for HLT (WLM): Will We Ever Really Replace the N-gram Model?*, pages 20–28, Stroudsburg, PA, USA, June 2012. Association for Computational Linguistics.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, May 2015.

[4] L. Bahl, P. Brown, P. de Souza, and R. Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *Proceedings of the 1986 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 49–52, Piscataway, NJ, USA, April 1986. IEEE.

[5] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, February 1970.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, February 2003.

[7] Yoshua Bengio and Jean-Sébastien Senécal. Quick training of probabilistic neural nets by importance sampling. In Christopher M. Bishop and Brendan J. Frey, editors, *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, New Jersey, USA, January 2003. Society for Artificial Intelligence and Statistics.

[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994.

[9] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, March 1996.

[10] Hervé Bourlard and Nelson Morgan. A continuous speech recognition system embedding MLP into HMM. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 186–193. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1990.

[11] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December 1992.

[12] Ivan Bulyko, Mari Ostendorf, and Andreas Stolcke. Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (HLT-NAACL): Short Papers*, volume 2, pages 7–9, Stroudsburg, PA, USA, May/June 2003. Association for Computational Linguistics.

[13] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–394, October 1999.

[14] Xie Chen, Xunying Liu, Mark J. F. Gales, and Philip C. Woodland. Recurrent neural network language model training with noise contrastive estimation for speech recognition. In *Proceedings of the 2015 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5411–5415, Piscataway, NJ, USA, April 2015. IEEE.

[15] Mathias Creutz, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Vesa Siivola, Matti Varjokallio, Ebru Arisoy, Murat Saraçlar, and Andreas Stolcke. Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM Transactions on Speech and Language Processing (TSLP)*, 5(1):3:1–3:29, December 2007.

[16] Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. In *Proceedings of the ACL 2002 Workshop on Morphological and Phonological Learning (MPL)*, volume 6, pages 21–30, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[17] Mathias Creutz and Krista Lagus. Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0. Report A81 in Publications in Computer and Information Science, Neural Networks Research Centre, Helsinki University of Technology, 2005.

[18] Mathias Creutz and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)*, 4(1):3:1–3:34, January 2007.

[19] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *Computing Research Repository*, abs/1612.08083, September 2017.

[20] K. H. Davis, R. Biddulph, and Stephen Balashek. Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6):637–642, November 1952.

[21] Steven B. Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, August 1980.

[22] Sabine Deligne and Frédéric Bimbot. Inference of variable-length linguistic and acoustic units by multigrams. *Speech Communication*, 23(3):223–241, November 1997.

[23] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, July 2011.

[24] Will J. Ebel and Joseph Picone. Human speech recognition performance on the 1994 CSR Spoke 10 corpus. In *Proceedings of the ARPA Spoken Language Systems Technology Workshop*, pages 53–59, January 1995.

[25] Michael Finke and Alex Waibel. Speaking mode dependent pronunciation modeling in large vocabulary conversational speech recognition. In *Proceedings of the 5th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 2379–2382. ISCA, September 1997.

[26] James W. Forgie and Carma D. Forgie. Results obtained from a vowel recognition computer program. *The Journal of the Acoustical Society of America*, 31(11):1480–1489, November 1959.

[27] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, October 2000.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[29] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., Red Hook, NY, USA, 2014.

[30] Joshua Goodman. Classes for fast maximum entropy training. In *Proceedings of the 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 561–564, Piscataway, NJ, USA, May 2001. IEEE.

[31] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 369–376, New York, NY, USA, June 2006. ACM.

[32] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML)*, volume 32 of *Proceedings of Machine Learning Research*, pages 1764–1772. PMLR, June 2014.

[33] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 297–304, New Jersey, USA, May 2010. Society for Artificial Intelligence and Statistics.

[34] Michael U. 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, February 2012.

[35] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012.

[36] Teemu Hirsimäki, Mathias Creutz, Vesa Siivola, and Mikko Kurimo. Morphologically motivated language models in speech recognition. In Timo Honkela, Ville Könönen, Matti Pöllä, and Olli Simula, editors, *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR)*, pages 121–126. Helsinki University of Technology, Laboratory of Computer and Information Science, June 2005.

[37] Teemu Hirsimäki, Mathias Creutz, Vesa Siivola, Mikko Kurimo, Sami Virpioja, and Janne Pylkkönen. Unlimited vocabulary speech recognition with morph language models applied to Finnish. *Computer Speech and Language*, 20(4):515–541, October 2006.

[38] Teemu Hirsimäki, Janne Pylkkönen, and Mikko Kurimo. Importance of high-order n-gram models in morph-based speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(4):724–732, 2009.

[39] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diplomarbeit, Technische Universität München, München, Germany, June 1991.

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

[41] Paria Jamshid Lou and Mark Johnson. Disfluency detection using a noisy channel model and a deep neural language model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 2 (Short Papers), pages 547–553, Stroudsburg, PA, USA, August 2017. Association for Computational Linguistics.

[42] Frederick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, April 1976.

[43] Frederick Jelinek and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In Edzard S. Gelsema and Laveen N. Kanal, editors, *Proceedings of the Workshop on Pattern Recognition in Practice*, pages 381–397, Amsterdam, The Netherlands, May 1980. North-Holland Publishing Company.

[44] Shihao Ji, S. V. N. Vishwanathan, Nadathur Satish, Michael J. Anderson, and Pradeep Dubey. Blackout: Speeding up recurrent neural network language models with very large vocabularies. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

[45] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(3):400–401, March 1987.

[46] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, May 2015.

[47] Dietrich Klakow. Log-linear interpolation of language models. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP)*. ISCA, December 1998.

[48] Dietrich Klakow. Selecting articles from the language model training corpus. In *Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 3, pages 1695–1698, Piscataway, NJ, USA, June 2000. IEEE.

[49] Reinhard Kneser and Hermann Ney. Forming word classes by statistical clustering for statistical language modelling. In Reinhard Köhler and BurghardB. Rieger, editors, *Contributions to Quantitative Linguistics*, pages 221–226. Kluwer Academic Publishers, Dordrecht, the Netherlands, 1993.

[50] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Proceedings of the 1995 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 181–184, Piscataway, NJ, USA, May 1995. IEEE.

[51] Oskar Kohonen, Sami Virpioja, and Krista Lagus. Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 78–86, Stroudsburg, PA, USA, July 2010. Association for Computational Linguistics.

[52] Mikko Kurimo, Antti Puurula, Ebru Arisoy, Vesa Siivola, Teemu Hirsimäki, Janne Pylkkönen, Tanel Alumäe, and Murat Saraclar. Unlimited vocabulary speech recognition for agglutinative languages. In *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pages 487–494, Stroudsburg, PA, USA, June 2006. Association for Computational Linguistics.

[53] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, April 1983.

[54] Sung-Chien Lin, Chi-Lung Tsai, Lee-Feng Chien, Keh-Jiann Chen, and Lin-Shan Lee. Chinese language model adaptation based on document classification and multiple domain-specific language models. In *Proceedings of the 5th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1463–1466. ISCA, September 1997.

[55] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, CA, USA, 1967. University of California Press.

[56] Milind Mahajan, Doug Beeferman, and X. D. Huang. Improved topic-dependent language modeling using information retrieval techniques. In *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 541–544, Piscataway, NJ, USA, March 1999. IEEE.

[57] Sven Martin, Jörg Liermann, and Hermann Ney. Algorithms for bigram and trigram word clustering. In *Proceedings of the 4th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1253–1256. ISCA, September 1995.

[58] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley series in probability and statistics. Wiley, Hoboken, NJ, second edition, 2008.

[59] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Coherent dialogue with attention-based language models. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, Palo Alto, CA, USA, February 2017. AAAI.

[60] Tomáš Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR) Workshops*, May 2013.

[61] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 1045–1048. ISCA, September 2010.

[62] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5528–5531, Piscataway, NJ, USA, May 2011. IEEE.

[63] Tomáš Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., Red Hook, NY, USA, 2013.

[64] Tomáš Mikolov, Scott Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT)*, pages 746–751, Stroudsburg, PA, USA, June 2013. Association for Computational Linguistics.

[65] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1751–1758, New York, NY, USA, July 2012. Omnipress.

[66] Robert C. Moore and William Lewis. Intelligent selection of language model training data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL) Conference Short Papers*, pages 220–224, Stroudsburg, PA, USA, July 2010. Association for Computational Linguistics.

[67] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, pages 246–252, New Jersey, USA, January 2005. Society for Artificial Intelligence and Statistics.

[68] A. Nádas. A decision theorectic formulation of a training problem in speech recognition and a comparison of training by unconditional versus conditional maximum likelihood. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(4):814–817, August 1983.

[69] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[70] Franz Josef Och. Maximum-Likelihood-Schätzung von Wortkategorien mit Verfahren der kombinatorischen Optimierung. Studienarbeit, Friedrich-Alexander-Universität Erlangen-Nürnburg, Erlangen, Germany, 1995.

[71] D. S. Pallett. A look at NIST's benchmark ASR tests: past, present, and future. In *Proceedings of the 2003 IEEE Workshop on Automatic Speech*

Recognition and Understanding (ASRU), pages 483–488. IEEE, November/December 2003.

[72] Junho Park, Xunying Liu, Mark J. F. Gales, and Philip C. Woodland. Improved neural network based language modelling and adaptation. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 1041–1044. ISCA, September 2010.

[73] Razvan Pascanu, Tomáš Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318. PMLR, June 2013.

[74] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi speech recognition toolkit. In *2011 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU) Demonstration Session*. IEEE Signal Processing Society, December 2011.

[75] Janne Pylkkönen. Investigations on discriminative training in large scale acoustic model estimation. In *Proceedings of the 10th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 220–223. ISCA, September 2009.

[76] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, Philadelphia, PA, USA, March 1998.

[77] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, September 1978.

[78] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, November 1997.

[79] Holger Schwenk and Jean-Luc Gauvain. Training neural network language models on very large corpora. In *Proceedings of the 2005 Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 201–208, Stroudsburg, PA, USA, October 2005. Association for Computational Linguistics.

[80] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Proceedings of the 12th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 437–440. ISCA, August 2011.

[81] Abhinav Sethy, Stanley F. Chen, Ebru Arisoy, and Bhuvana Ramabhadran. Unnormalized exponential and neural network language models. In *Proceedings of the 2015 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5416–5420, Piscataway, NJ, USA, April 2015. IEEE.

[82] Abhinav Sethy, Panayiotis G. Georgiou, and Shrikanth Narayanan. Text data acquisition for domain-specific language models. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*

(EMNLP), pages 382–389, Stroudsburg, PA, USA, July 2006. Association for Computational Linguistics.

[83] Y. Shi, W. Q. Zhang, M. Cai, and J. Liu. Variance regularization of RNNLM for speech recognition. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4893–4897, Piscataway, NJ, USA, May 2014. IEEE.

[84] Vesa Siivola, Teemu Hirsimäki, Mathias Creutz, and Mikko Kurimo. Unlimited vocabulary speech recognition based on morphs discovered in an unsupervised manner. In *Proceedings of the 8th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 2293–2296. ISCA, September 2003.

[85] Vesa Siivola and Bryan L. Pellom. Growing an n-gram language model. In *Proceedings of the 9th European Conference on Speech Communication and Technology (INTERSPEECH – EUROSPEECH)*, pages 1309–1312. ISCA, September 2005.

[86] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[87] Rupesh K Srivastava, Klaus Greff, and Juergen Schmidhuber. Training very deep networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2377–2385. Curran Associates, Inc., Red Hook, NY, USA, 2015.

[88] Andreas Stolcke. Entropy-based pruning of backoff language models. In *Proceedings of the Broadcast News Transcription and Understanding Workshop*, pages 270–274. Morgan Kaufmann Publishers, February 1998.

[89] Andreas Stolcke. SRILM — an extensible language modeling toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP)*, pages 901–904, September 2002.

[90] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lattice decoding and rescoring with long-span neural network language models. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 661–665. ISCA, September 2014.

[91] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., Red Hook, NY, USA, 2014.

[92] Matti Varjokallio, Mikko Kurimo, and Sami Virpioja. Learning a subword vocabulary based on unigram likelihood. In *Proceedings of the 2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, December 2013.

[93] V.M. Velichko and N.G. Zagoruyko. Automatic recognition of 200 words. *International Journal of Man-Machine Studies*, 2(3):223–234, July 1970.

[94] Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Python implementation and extensions for Morfessor Baseline. Report 25/2013 in Aalto University publication series SCIENCE + TECHNOLOGY, Department of Signal Processing and Acoustics, Aalto University, 2013.

[95] Philip C. Woodland and Daniel Povey. Large scale discriminative training of hidden Markov models for speech recognition. *Compututer Speech and Language*, 16(1):25–47, January 2002.

[96] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. Technical report, Microsoft Research Lab, Redmond, WA, USA, February 2017.

[97] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *Computing Research Repository*, abs/1212.5701, December 2012.

[98] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Proceedings of the 13th European Conference on Computer Vision (ECCV)*, pages 818–833, Cham, Switzerland, September 2014. Springer International Publishing.

[99] Yizhe Zhang, Zhe Gan, and Lawrence Carin. Generating text via adversarial training. In *Proceedings of the NIPS Workshop on Adversarial Training*, December 2016.

Errata

Publication I

Equation 2 has incorrect signs in the final publication. The correct form is $-g_i(X) + log \sum_{j \neq i} \exp(g_j(X))$.

Publication III

The Finnish NNLM models were based on subword units and the Estonian on compound-split words, while few sentences in the published paper in Section 3.4 erroneously claim the opposite.

Publication IV

Kneser–Ney smoothing was not used when training the class-based n-gram models, because the class n-gram statistics were not suitable for the Modified Kneser–Ney implementation. While this is not said in the article, the effect of smoothing is not that significant in class-based models, because the data is not as sparse.

Publication V

A highway network layer uses a separate bias b_σ for its gate. The index σ is missing from Equation 6 in the published paper. The correct form is $g(x_t) = \sigma(W_\sigma x_t + b_\sigma)$.

Automatic recognition of Finnish speech has been developed for decades, and very low error rates have been achieved on clearly spoken standard Finnish, such as news broadcasts. Recognition of natural conversations is much more challenging. The language that is used in Finnish conversations also differs in many ways from standard Finnish, and its recognition requires data that has previously been unavailable.

This thesis develops automatic speech recognition for conversational Finnish, starting from data collection. For language modeling, large amounts of text are collected from the Internet, and filtered to match the colloquial speaking style. An evaluation set is published and used to benchmark the progress in conversational Finnish speech recognition. The thesis addresses many difficulties that arise from the fact that the vocabulary that is used in Finnish conversations is very large. By modeling speech and language using artificial neural networks, accuracy that is already useful for practical applications is achieved.

Aalto-DD 52/2018

ISBN 978-952-60-7907-3 (printed)
ISBN 978-952-60-7908-0 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Electrical Engineering
Department of Signal Processing and Acoustics
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS