Tutorial M3:
Building an Open Vocabulary ASR System using Open Source Software

Stefan Hahn, David Rybach

`rwthasr@i6.informatik.rwth-aachen.de`

Interspeech 2011, Florence
27/08/2011

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6
Computer Science Department
RWTH Aachen University, Germany

Outline

- Introduction and Goals
- Acoustic Model Training using R*ASR*
- Vocabulary Selection and Pronunciation Generation
- Language Model Training using SRI LM
- Decoding using R*ASR* and Evaluation
- Open Vocabulary Recognition and Evaluation
- Outlook

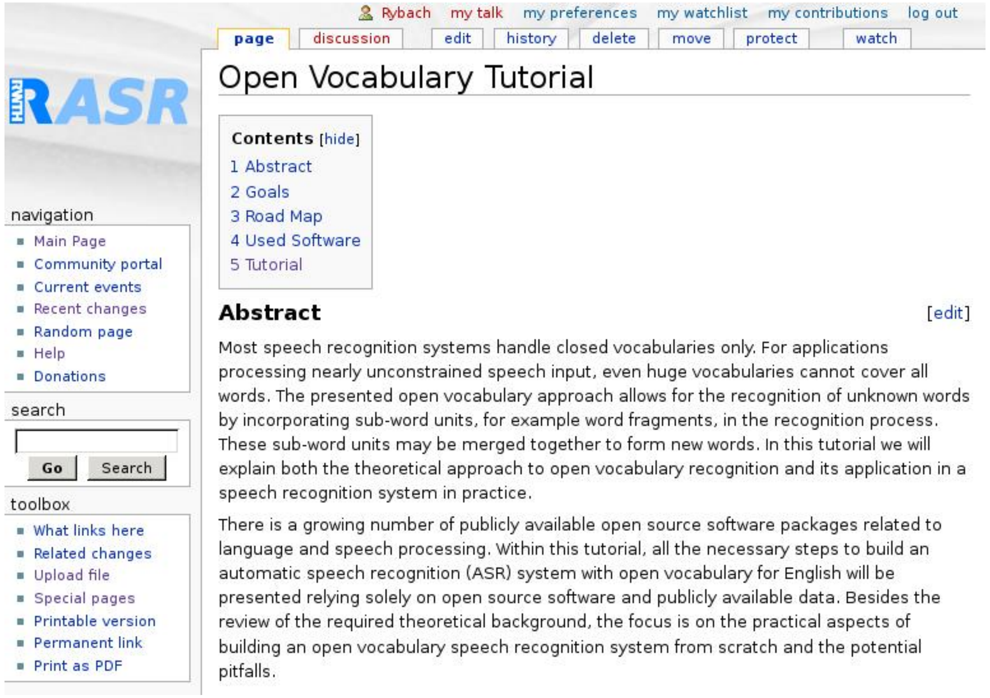
Introduction

- common ASR systems: closed vocabulary
- unconstrained speech data requires huge vocabulary
- even then not all words are covered
- open vocabulary recognition: allow for the recognition of sub-word units

this tutorial:

- practical aspects of ASR system development
- open vocabulary approach using word fragments
- only open source software and publicly available data
- small example task: system can be build on your computer
- all scripts and configuration files are available

Build Your Own ASR System



Rybach my talk my preferences my watchlist my contributions log out

page discussion edit history delete move protect watch

RASR

navigation
- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

search

Go Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Print as PDF

Open Vocabulary Tutorial

Contents [hide]

1 Abstract
2 Goals
3 Road Map
4 Used Software
5 Tutorial

Abstract [edit]

Most speech recognition systems handle closed vocabularies only. For applications processing nearly unconstrained speech input, even huge vocabularies cannot cover all words. The presented open vocabulary approach allows for the recognition of unknown words by incorporating sub-word units, for example word fragments, in the recognition process. These sub-word units may be merged together to form new words. In this tutorial we will explain both the theoretical approach to open vocabulary recognition and its application in a speech recognition system in practice.

There is a growing number of publicly available open source software packages related to language and speech processing. Within this tutorial, all the necessary steps to build an automatic speech recognition (ASR) system with open vocabulary for English will be presented relying solely on open source software and publicly available data. Besides the review of the required theoretical background, the focus is on the practical aspects of building an open vocabulary speech recognition system from scratch and the potential pitfalls.

- **step-by-step tutorial is available online**
  http://www.hltpr.rwth-aachen.de/rwth-asr/manual
- **requires basic knowledge of UNIX/Linux command line tools**
- **scripts are mostly written in Python and Bash**
- **all data and software used is publicly available and free**

Used Software

- R*ASR* V0.5 (RWTH Aachen University)
  `http://www.hltpr.rwth-aachen.de/rwth-asr`
- SRI LM Toolkit V1.5.11 (SRI International)
  `http://www-speech.sri.com/projects/srilm/download.html`
- Sequitur G2P r.1668 (RWTH Aachen University)
  `http://www.hltpr.rwth-aachen.de/web/Software/g2p.html`
- SCTK Scoring Package V2.4.0 (NIST)
  `http://www.itl.nist.gov/iad/mig/tools/`
- Morfessor 1.0 (Helsinki University of Technology)
  `http://www.cis.hut.fi/projects/morpho/`
- installation instructions in the Wiki

Goals

learn how to

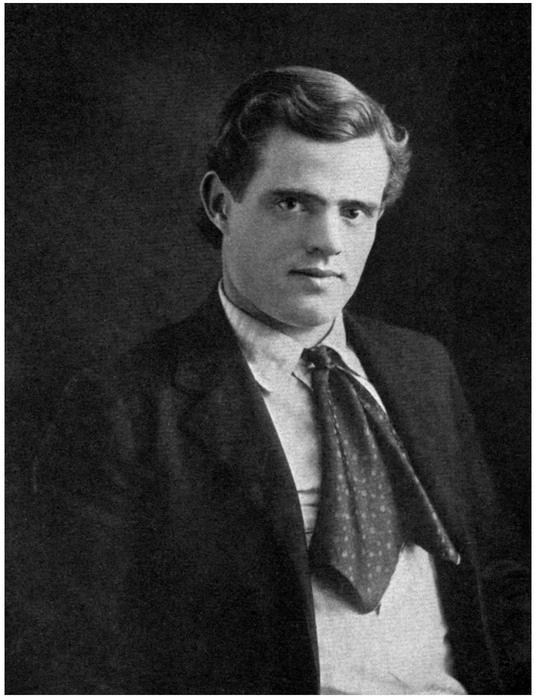
- build acoustic models using R*ASR*
- estimate language models using SRI LM
- generate pronunciation models using Sequitur G2P
- run R*ASR*'s speech recognition engine
- evaluate ASR systems with NIST SCLITE
- build an open vocabulary ASR system

Audio Data: Voxforge

- http://www.voxforge.org/ offers
  - free GPL speech audio in various languages
  - accompanying transcriptions
  - lexica for recognition as well as speech synthesis
- CMU ARCTIC Corpus
  - read English speech (novels)
  - several speakers: male/female, various dialects
  - repeated sentences (once per speaker)
  - examples:
    - awbarcticb0404.wav
    - clbarcticb0404.wav
    - jmkarcticb0393.wav

Text Data: Project Gutenberg

- http://www.gutenberg.org/ offers
  - free to use eBooks in various languages
  - drawback: mostly “old” texts, not well formatted for computational processing



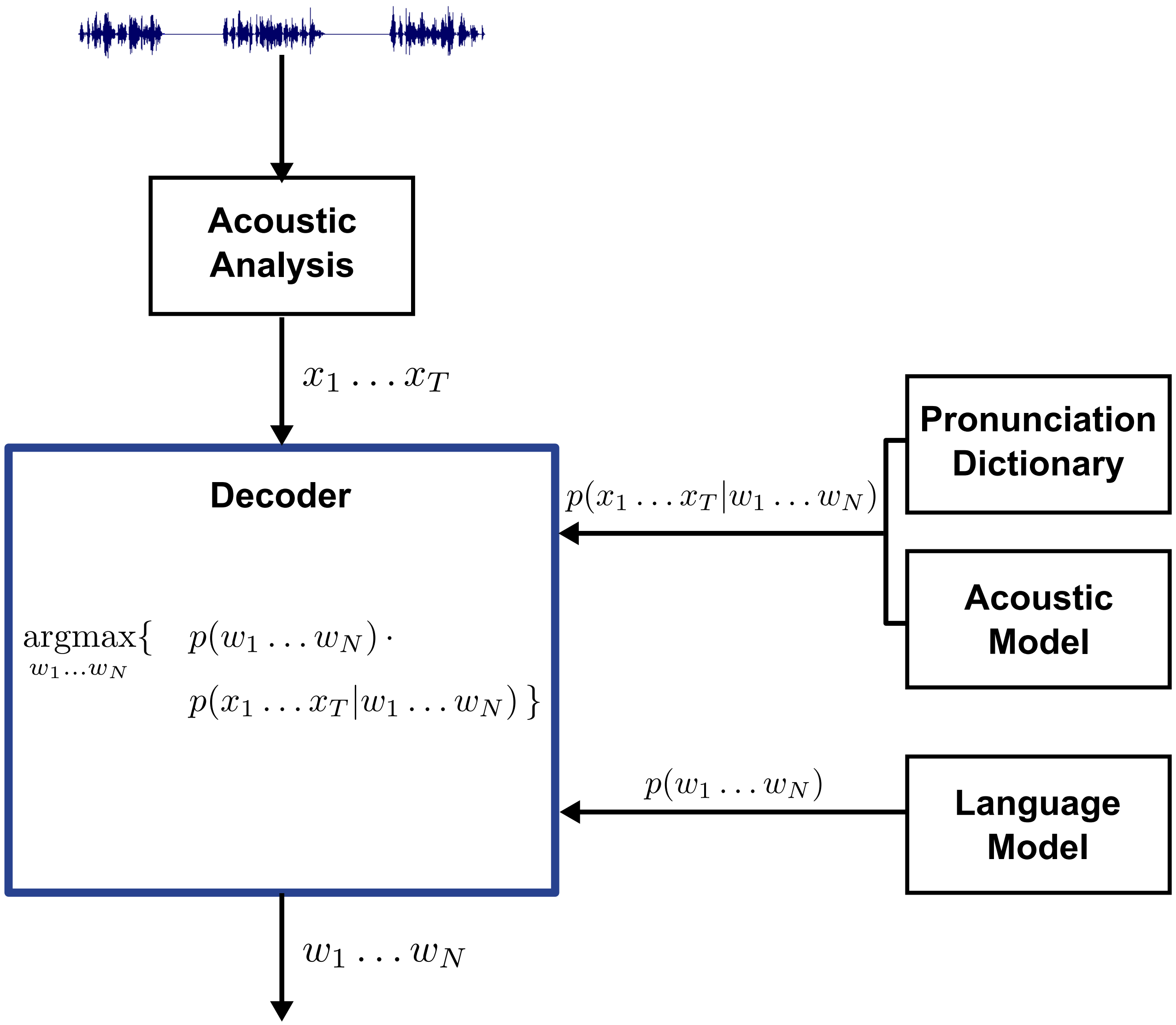
Jack London

Gilbert Parker

Edward Bulwer Lytton

Outline

- Introduction and Goals
- Acoustic Model Training using R*ASR*
- Vocabulary Selection and Pronunciation Generation
- Language Model Training using SRI LM
- Decoding using R*ASR* and Evaluation
- Open Vocabulary Recognition and Evaluation
- Outlook


Acoustic Analysis
$x_1 \ldots x_T$
Decoder
$\underset{w_1 \ldots w_N}{\operatorname{argmax}} \{ \; p(w_1 \ldots w_N) \cdot p(x_1 \ldots x_T | w_1 \ldots w_N) \}$
$p(x_1 \ldots x_T | w_1 \ldots w_N)$
Pronunciation Dictionary
Acoustic Model
$p(w_1 \ldots w_N)$
Language Model
$w_1 \ldots w_N$

Speech Recognition: Knowledge Sources

pronunciation dictionary

- map words to sequences of phonemes
- generated by linguists and/or automatically
- automatic grapheme to phoneme (G2P) conversion: requires additional (statistical) model

acoustic model (AM)

- statistical models for acoustic realization of words / phonemes
- model (context dependent) phones by HMMs
- Gaussian mixture model as HMM state emission model

language model (LM)

- statistical model of syntax and semantics
- commonly used: simple n-gram model

open vocabulary (OV) models

- word fragmentation: split words into sub-words
- hybrid LM: sequences of words and fragments

R*ASR*

- RWTH Aachen University Open Source Speech Recognition Toolkit
- short: RWTH ASR, shorter: R*ASR*
- developed by the Human Language Technology and Pattern Recognition Group at RWTH Aachen University
- framework for all ASR research topics and projects
- very flexible and extendable
- framework also used for sign language recognition, OCR, video/image processing
- several libraries and tools written in C++
- supported platforms: Linux (x86), Mac OS X
- documentation: `http://www.hltpr.rwth-aachen.de/rwth-asr/manual`

`http://www.hltpr.rwth-aachen.de/rwth-asr`

[Rybach & Gollan+ 09]

R*ASR*: Features

- LVCSR decoder
- flexible signal processing framework: Flow
- variable acoustic modeling
- language modeling: ARPA, weighted grammar
- speaker adaptation / normalization
- acoustic model training (ML, MPE)
- interfaces to math libraries (LAPACK, BLAS)
- lattice processing

RASR: Configuration

- components receive their configuration from a global configuration instance
- the configuration is set up from configuration files and command line arguments
- a configuration resource is a key-value pair
- keys have the form `<selector1>. ... .<selectorN>.<parameter>`
- each selector corresponds to a component name
- components are organized hierarchically
- selectors can be replaced by the wildcard `*`

- example:

```
*.corpus.file                            = recognition.corpus
*.acoustic-model.hmm.states-per-phone = 3
```

```
speech-recognizer.linux-intel-standard --config=recognition.config \
                                       --*.corpus.file=other.corpus
```

RASR: Configuration

- configuration resources with equal selectors can be grouped
- example:

```
[*.acoustic-model.hmm]
states-per-phone               = 3
state-repetitions              = 2
```

- configuration values can include **references**: `$(selector)`
- the reference is textually replaced by the resolved value
- example:

```
DATA_DIR = /var/tmp/intermediate
TASK_ID = 0004
feature-cache.path           = $(DATA_DIR)/features.$(TASK_ID)
```

- configuration files can include other files using the `include` directive
- example:

```
include config/shared.config
```

Configuration Examples

training/config/feature-extraction.mfcc.config

```
DESCRIPTION = feature-extraction.mfcc

include shared.config
include channels.config
include corpus.config

[*.corpus]
file                                = $(TRAIN_CORPUS)

[*.feature-extraction]
*.network-file-path                 = $(FLOW_DIR)
file                                = $(FLOW_DIR)/shared/base.cache.flow
*.base-feature-extraction.file      = mfcc/mfcc.postprocessing.flow
*.audio-format                      = wav

[*.base-feature-extraction-cache]
path                                = $(DATA_DIR)/mfcc.features.cache
```

training/config/shared.config

```
[*]
CORPUS_DIR = /u/corpora/speech/voxforge
FLOW_DIR = $(RESOURCES)/rwth-asr/flow
# ...
TRAIN_CORPUS = $(CORPUS_DIR)/training_clb_rms_awb.corpus.gz
```

RASR: Channels

- components use channels for messages / text output
- the content written to a channel is sent to its targets
- each channel can be configured and redirected separately
- configuration of channels and targets is done using the standard configuration process
- channel targets can be either predefined targets (`stdout`, `stderr`, `nil`) or custom targets defined by the configuration

```
[*]
log.channel                  = output-channel
warning.channel              = output-channel, stderr
error.channel                = output-channel, stderr
dot.channel                  = nil

recognizer.statistics.channel = output-channel

[*.channels.output-channel]
file             = log/my-logfile.log.gz
append           = false
encoding         = UTF-8
unbuffered       = false
compressed       = true
```

RASR: Corpus

- audio files
- segmentation
- unique identifiers for segments and recordings
- audio / segment meta data (speaker, recording condition, ...)
- transcriptions

```
<corpus name="example">
    <speaker-description name="peter">
        <gender> male </gender>
    </speaker-description>
    <recording audio="example_db/peter/file_01.wav" name="rec_01">
        &lt;segment start="10.50" end="12.22" name="s_04">
            <!-- segment ID: example/rec_01/s_04 -->
            <speaker name="peter"/>
            <orth>
                hello my name is peter
            </orth>
        &lt;/segment&gt;
    </recording>
</corpus>
```

Generating Corpus Files (1)

- split data by utterance id in training and test set
- make sure sentences do not occur in both sets
- usually: separate development set for parameter tuning
- here: database is small - no development set
- quick AM training: limit corpus to 4 speakers (bdl, slt, clb, rms)
- segmentation is provided (one utterance per recording)
- some words require normalization

  `75` → `seventy five`

  `e.g.` → `for example`

	training	test
segments	4,072	456
duration [h:mm]	3:34	0:24
words	36,156	4,108

Generating Corpus Files (2)

`raw/voxforge/cmu_us_awb_arctic/etc/txt.done.data`

```
( arctic_a0032 "Since then some mysterious force has been fighting us at every step." )
( arctic_a0033 "He unfolded a long typewritten letter, and handed it to Gregson." )
```

- **transform raw database to R*ASR* format: `scripts/gencorpus.py`**

`xml/training_bdl_slt_clb_rms.corpus`

```
<corpus name="CMU_ARCTIC">
  <recording audio="cmu_us_awb_arctic/wav/arctic_a0032.wav" name="awb_a0032">
    &lt;segment start="0" end="inf"&gt;
      <speaker name="awb"/>
      <orth>since then some mysterious force has been fighting us at every step</orth>
    &lt;/segment&gt;
  </recording>
  <recording audio="cmu_us_awb_arctic/wav/arctic_a0033.wav" name="awb_a0033">
    &lt;segment start="0" end="inf"&gt;
      <speaker name="awb"/>
      <orth>he unfolded a long typewritten letter and handed it to gregson</orth>
    &lt;/segment&gt;
```

RASR: Lexicon (1)

- **the lexicon defines the phoneme set and the pronunciation dictionary**
- **the basic unit of the lexicon is a lemma**

```
<lemma>
  <orth>the</orth>
  <phon>dh ax</phon>
  <phon>dh ah</phon>
  <phon>dh iy</phon>
</lemma>
```

- **a lemma can consist of**
  - **one or more orthographic forms (written word)**
  - **any number of pronunciations (phonetic transcriptions)**
  - **a sequence of syntactic tokens (language model tokens)**
  - **a sequence of evaluation tokens (word used for evaluation)**

RASR: Lexicon (2)

- special lemmata define properties of silence, sentence begin/end, unknown words
- silence lemma: add empty orthography to hypothesize silence between words (in training):

```
<lemma special="silence">
  <!-- preferred orthographic form -->
  <orth>[SILENCE]</orth>
  <!-- empty orthography: hypothesize between all words -->
  <orth/>
  <phon>si</phon>
  <!-- empty syntactic token sequence: no LM token -->
  <synt/>
  <!-- empty evaluation token sequence: ignore for WER computation -->
  <eval/>
</lemma>
```

Digression: Why is ASR using phonemes? (1)

- this is not the reason: phonemes describe the sounds of the language
- purpose of using phonemes in ASR: state tying
  - a “phoneme” is just a label for an HMM
  - more typically: a predicate in the state tying decision tree

- Can we just use the letters of the written form, and leave the rest to acoustic modeling?
  - Yes: works for words that have normal pronunciations or that are very frequent
  - No: long tail of infrequent words, some of which have strange pronunciations

[Kanthak & Ney 03]

Digression: Why is ASR using phonemes? (2)

- a practical system needs a mapping layer between written and spoken forms to cope with irregularities in this relation:
  - Winchester ↔ Worchester
  - Ke$ha
  - A. sending ↔ ascending
- linguists know how to describe pronunciations using phonemes (more often than not) consistently and unambiguously

Create Training Lexicon

raw/VoxForgeDict

```
THE              [THE]            dh ax
THE(2)           [THE]            dh ah
THE(3)           [THE]            dh iy
THEA             [THEA]           th iy ax
```

- convert to RASR format: `scripts/genlexicon.py`

xml/training.lexicon

```
<lexicon>
  <phoneme-inventory>
    <phoneme>
      <symbol>aa</symbol>
    </phoneme>
    <!-- .... -->
  </phoneme-inventory>
  <lemma special="silence">
    <orth>[SILENCE]</orth>
    <orth/>
    <phon>si</phon>
    <synt/>
    <eval/>
  </lemma>
  <lemma>
    <orth>fawn</orth>
    <phon>f ao n</phon>
  </lemma>
```

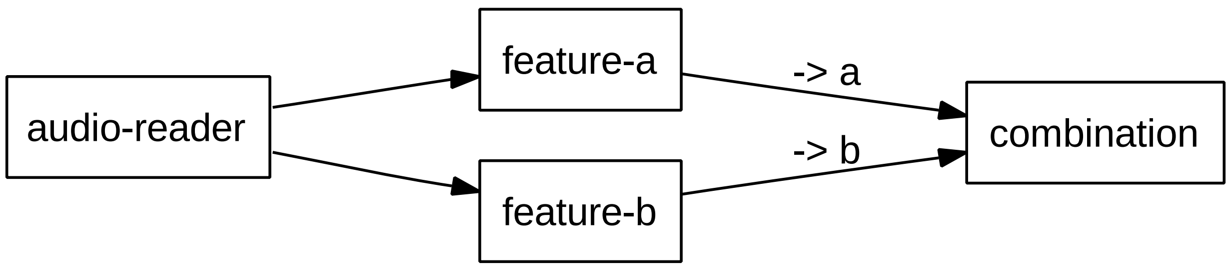
Compute Corpus Statistics

- **R*ASR* tool:** `costa`

`xml/log/test_bdl_slt_clb_rms.costa.log`

```
<size-statistics>
  number of recordings                         : 342
  number of speech segments                    : 342
  number of other segments                     : 0
  gross duration (all recordings)              : 1269.14s
  net duration (all segments)                  : infs
  net number of time frames (all segments)     : 0
</size-statistics>
<!-- ... -->
<lexical-statistics>
  number of orthographic match positions               : 3066
  number of orthographic match failures                : 3
  number of matched lemmas                             : 6471
  number of non-loop matched lemmas                    : 3063
  number of matched pronunciations                     : 8239
  number of non-loop matched pronunciations            : 4831
  number of matched evaluation token sequences         : 6471
  number of non-loop matched eval token sequences      : 3063
  avg. number of matched evaluation tokens             : 3063
  avg. number of non-loop matched eval tokens          : 3063
  avg. number of phonemes (average of variants)        : 14168.6
  <!-- occurances of known words -->
  <lemma name="wednesday" count="2">
    <orth>wednesday</orth>
  </lemma>
```

- flow is a general framework for data processing
- currently it is mainly used for feature extraction and alignment generation
- data manipulation (including loading / storing) is done by **nodes**
- a node has zero or more input and output **ports**
- nodes are connected by **links** from one port to another
- a set of nodes combined by links forms a **network**
- a network can be used as node itself
- flow networks are defined in XML documents
- nodes are either predefined in the software or another flow file
- abstract example:



RASR: Flow

```
<network>
  <out name="features"/>
  <param name="input-file"/>
  <param name="start-time"/>
  <param name="end-time"/>

  <node name="audio-reader" filter="audio-input-file-wav"
        file="$(input-file)" start-time="$(start-time)" end-time="$(end-time)"/>

  <node name="mfcc" filter="mfcc.flow"/>
  <link from="audio-reader" to="mfcc"/>

  <node name="lda" filter="lda.flow"/>
  <link from="mfcc" to="lda:in"/>

  <link from="lda:out" to="network:features"/>
</network>
```

RASR: Flow Caches

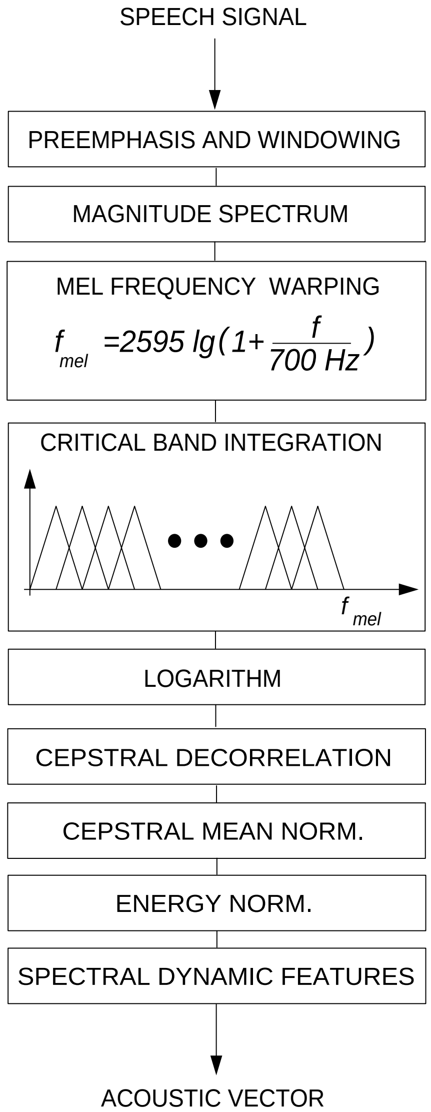
- all data sent in a flow network can be stored in caches
- a cache node has one input and one output port
- data requested at the output port of a cache node:
  - first check if the data is present in the cache
  - if not, the data is requested at the node's input port (if connected)
  - and stored in the cache
- cached items are identified by the parameter `id`
- segment ids are propagated by the network
- caches are used primarily to store features and alignments

```
<node name="base-feature-extraction" filter="$(file)"/>

<node name="base-feature-cache" filter="generic-cache" id="$(id)"/>
<link from="base-feature-extraction" to="base-feature-cache"/>

<link from="base-feature-cache" to="network:features"/>
```

RASR: MFCC Features

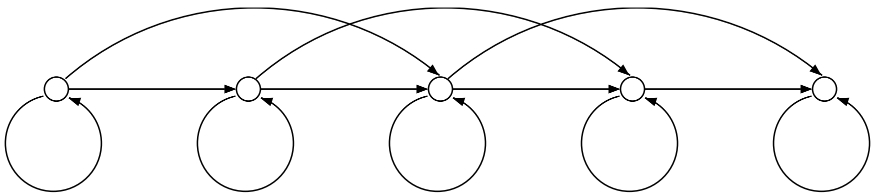
- 16 MFCCs
- segment-wise cepstral mean normalization
- incorporation of temporal context: use temporal derivatives
- features are pre-computed and stored in Flow cache files

- `flow/shared/base.cache.flow`:
  read samples, extract features, store in cache
- `flow/mfcc/mfcc.postprocessing.flow`:
  compute MFCC features
- `flow/shared/concatenate-with-derivatives.flow`:
  compute and concatenate derivatives



[Bridle & Brown 74, Mermelstein 76]

R*ASR*: Hidden Markov Models (1)



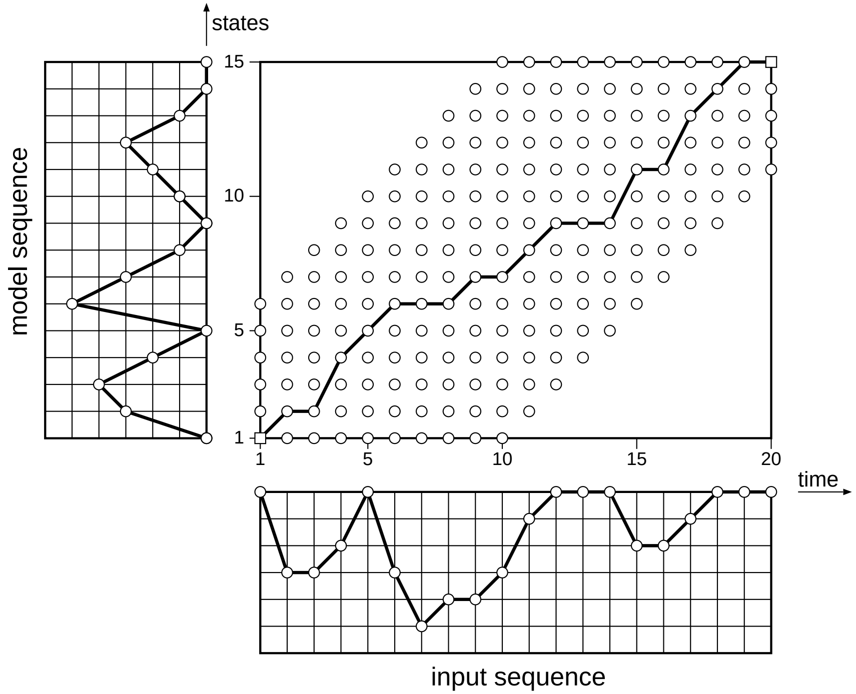
- common HMM topology for a phoneme: 3 states
- parameters: `hmm.states-per-phone`, `hmm.state-repetitions`
- transition probabilities are defined state independently for loop, forward, skip, and exit transitions (`tdp`)
- silence phoneme
  - only 1 state (`tdp.silence`)

RASR: Hidden Markov Models (2)

training/config/hmm.config

```
[*.acoustic-model.hmm]
states-per-phone                = 3
state-repetitions               = 1
across-word-model               = no
early-recombination             = no

[*.acoustic-model.tdp]
scale                           = 1.0
*.loop                          = 3.0
*.forward                       = 0.0
*.skip                          = infinity
*.exit                          = 0.0
entry-m1.loop                   = infinity
entry-m2.loop                   = infinity

silence.loop                    = 0.0
silence.forward                 = 3.0
silence.skip                    = infinity
silence.exit                    = 3.0
```

RASR: Alignment



- **non-linear time alignment:**
  align sequence of feature vectors to sequence of HMM states
- **Viterbi algorithm**
  $\Rightarrow$ each feature vector is mapped to exactly one HMM state
- **alignment is generated by a Flow node `speech-alignment`**
- **alignments can be stored in Flow caches**

- **HMM state emission model: Gaussian mixture model**

$$p(x|s) = \sum_{l=1}^{L(s)} c_{ls} \cdot \det(2\pi\Sigma_{ls})^{-\frac{1}{2}} \cdot \exp\left(-\frac{1}{2}(x-\mu_{ls})^T \Sigma_{ls}^{-1} (x-\mu_{ls})\right)$$

- **options for covariance modeling**
  - **mixture specific covariance:** $\Sigma_{ls} = \Sigma_s \; \forall l$
  - **globally pooled covariance:** $\Sigma_{ls} = \Sigma \; \forall l, s$
  - **diagonal covariance:** $\Sigma_{ls} = I \cdot \sigma_{ls}^2$
- **R*ASR* default model**
  - **globally pooled diagonal covariance**
  - **similar number of parameters by using more mean vectors**

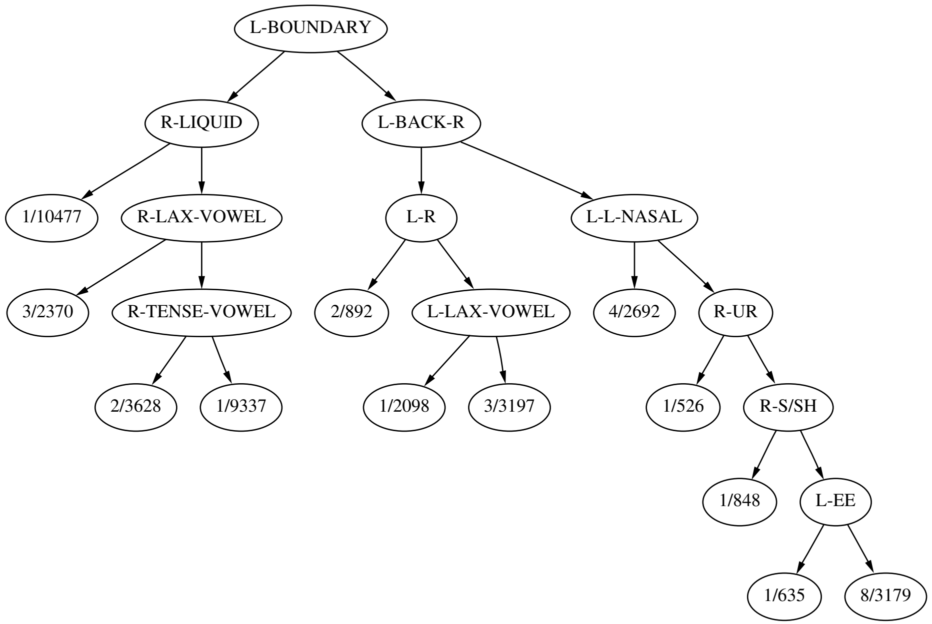
RASR: Mixture Model Training

- here: maximum likelihood training
- iterative optimization
- initialization: linear segmentation (alignment without AM)

- generate alignment using previous model: assign x_t to HMM state (s, w)
- assign each x_t to a density (l, s, w)
- collect sufficient statistics for each density (“accumulate”)
- estimate new model
- optional: split density
- keep state alignment fixed for n accumulations / complete training

`training/monophone_model.sh`

- model phonemes in context: triphones
- problem: not enough data to train models for all triphone states ($3 \cdot |\text{phon.}|^3$)
- Classification and Regression Tree (CART)
- tie parameters of “similar” triphones using a phonetic decision tree
- node: split by asking phonetic questions
- leaves: generalized triphone HMM state model



[Young & Odell+ 94, Beulen & Bransch+ 97]

RASR: CART

- estimate CART for triphone HMM state tying
- training/config/cart-questions.xml:
  phonetic questions, cf. http://en.wikipedia.org/wiki/Arpabet

```
<for-each-key keys="history[0] future[0]">
    <for-each-value>
        <question description="context-phone"/>
    </for-each-value>
    <question description="vowel">
       <values> ao aa iy uw eh ih uh ah ae </values>
    </question>
    <question description="diphthongs">
        <values> ey ay ow aw oy </values>
    </question>
    <question description="stops">
        <values> p b t d k g </values>
    </question>
    <question description="affricates">
        <values> ch jh </values>
    </question>
```

- collect statistics for each triphone HMM state
- estimate CART
- training for triphone models

Outline

- Introduction and Goals
- Acoustic Model Training using R*ASR*
- Vocabulary Selection and Pronunciation Generation
- Language Model Training using SRI LM
- Decoding using R*ASR* and Evaluation
- Open Vocabulary Recognition and Evaluation
- Outlook

Lexicon Modeling

- distinction:
  - training: all words/pronunciations which occur in the training corpus, maybe even broken words
  - recognition: most frequent (orthographic correct) words
- how to select words for the recognition lexicon?
  - use (task dependent) text sources, the larger the better
  - vocabulary: frequency count statistics after preprocessing/normalization
  - (normalized data can be used directly for language modeling)
  - sources for pronunciations: dictionary, g2p or linguist
  - some of the selected words may not make sense and are filtered out in the pronunciation generating step

Vocabulary Selection and Language Modeling Data

- large text sources are used for vocabulary selection and LM training
- typically harvested off the internet or news-archives etc.
- text data for our task:
  - selection of Project Gutenberg texts
  - audio data transcripts
- for fair comparison: make sure that material of test set is not included
- preprocessing necessary to
  - normalize tokens
  - segment text into sentences for LM training
- text example:

```
"Let us continue on, then," I replied.  "It should soon be over at this
rate.  You never intimated that the speed of this thing would be so
high, Perry.  Didn't you know it?"
```

`gutenberg/james_oliver_curwood-The_Rivers_End.txt`

Text Preprocessing (1)

- normalize tokens to get good probabilities and count-statistics
- i.e. do not distribute the probability mass over various forms of the same token
  [Mister] [mr.] [“mister”] ["mister"] [mister,] -> [mister]
- common techniques (language/text dependent):
  - remove brackets, double quotes, non-word characters like ()[]"*_ etc
  - compress multiple dots: ... $\longrightarrow$.
  - (possibly) normalize numbers: 1999 -> nineteen ninety-nine
  - detach and discard sentence end symbols like .:;!? from words
  - expand abbreviations
  - lower case
- re-segment the text to have one sentence per line
- sentence-start and sentence-end symbols <s> and </s>

Text Preprocessing (2)

- first: Project Gutenberg files are preprocessed
- `scripts/preprocessGutenberg.py`
- the num2word library is used to normalize numbers
- the script is used to normalize all text files
- normalized text example:

```
<s> let us continue on then i replied </s>
<s> it should soon be over at this rate </s>
<s> you never intimated that the speed of this thing would be so high perry </s>
<s> didn't you know it </s>
```

Text Preprocessing (3)

- second: the transcriptions of the acoustic material are pre-processed:

  `xml/training_bdl_slt_clb_rms.corpus`

  `xml/test_bdl_slt_clb_rms.corpus`

- all `<orth>` lines are selected and saved in the format

  `<s> sentence </s>`

- since many sentences occur more than once (from several speakers), just `uniq` lines are selected

- target filenames:

  `data/lm/train.uniq.text`

  `data/lm/test.uniq.text`

Vocabulary (Pre-)Selection

- all normalized Gutenberg text files are merged into one file
- we will call the resulting file `background.text`
- the tokens of the normalized LM training texts are merged (`data/lm/background.text` and `data/lm/train.uniq.text`) and sorted w.r.t. word frequencies
- total counts are saved into separate files
- the most frequent 3k, 6k and 12k words resp. are selected as vocabularies, e.g. for 3k:
  `data/lm/vocab.3k`

why pre-selection?:

- words not making any sense may be included in the vocabularies (e.g. due to suboptimal text normalization)
- the g2p model may fail to translate certain words (will be explained later)
- thus: some words may be removed for the final vocabulary

Pronunciation Generation

- goal: generate pronunciations for the various vocabularies
- idea:
  - data source: `data/voxforge.dict`
  - if a word is not in the dictionary: use statistical model for pronunciation generation
  - therefore: utilize the Sequitur G2P software written by Max Bisani
- steps:
  - data preprocessing
  - train a g2p model
  - apply a g2p model to the words of the vocabularies which are not in the base dictionary
  - discard words which could not be processed by the g2p tool
  - convert word/pronunciation pairs to R*ASR* lexicon format
- result: recognition lexica for 3k, 6k and 12k vocabularies

Digression: Why not just using a dictionary?

- small vocabulary systems (e.g. digit recognition)
  train individual acoustic models for each word
- large vocabulary systems operate with a fixed large but finite vocabulary (vocabulary size typically 10k-100k word forms)
  - suitable for dictation tasks for a fixed domain, but less suitable for open vocabulary settings (e.g. broadcast news, podcasts)
  - the number of different words does not seem to be finite
  - the important content words change over time
  - not all words are known beforehand
- grapheme-to-phoneme conversion is needed to generalize beyond a fixed set of words

Data Preprocessing

- `raw/VoxForgeDict`
- preprocessing steps:
  - convert to lower case
  - remove tokens containing `0-9._`
  - remove possible abbreviations, i.e. `len(word)<4`
- a filtered version of the base dictionary is generated for g2p training
- for vocabulary look-up, a lower-case (un-filtered) version of the base dictionary is generated: `data/g2p/basis-dictionary.lower`
- the filtered dictionary is split into a training (`data/g2p/train.prondict`) and a development set (`data/g2p/dev.prondict`)

Grapheme-to-Phoneme Conversion

- **grapheme: a symbol used for writing language (e.g. a letter)**
- **phoneme : smallest contrastive unit in the sound system of a language**

- **given: orthographic form (grapheme sequence) $g \in G^*$**
- **task: find the most likely pronunciation (phoneme sequence) $\varphi \in \Phi^*$:**

$$\varphi(g) = \operatorname*{argmax}_{\varphi' \in \Phi^*} p(g, \varphi')$$

- **here: joint-n-gram approach as presented in [Bisani & Ney 08]**

[Deligne & Yvon$^+$ 95, Bisani & Ney 08, Jiampojamarn & Kondrak 09]

Evaluation Criteria

Performance Measures

- **phoneme error rate (PER)**
  Levenshtein distance divided by the number of phonemes in the reference pronunciation
- **word error rate (WER)**
  fraction of words containing at least one error

Pronunciation Variants

- **in some lexica, more than one reference pronunciation is given**
- **hypothesis is considered correct if it equals one of the given variants**

Graphones

- for a given orthographic form $g \in G^*$,
  what are the likely pronunciations $\varphi \in \Phi^*$?
- Assumption: for each word, its orthographic form and its pronunciation are generated by a common sequence of blocks that carry both letters and phonemes
- such a block is called grapheme-phoneme, joint-multigram or graphone for short
- formally, a graphone is a pair of a letter sequence and a phoneme sequence of possibly different length [Deligne & Yvon^{+} 95]

$$q = (g, \varphi) \in Q \subseteq G^* \times \Phi^*$$

- Example: sequence of six graphones

“mixing”	=	m	i	x	i	n	g
[mɪksɪŋ]		[m]	[ɪ]	[ks]	[ɪ]	[ŋ]	—

Joint Sequence Model

- joint distribution $p(g, \varphi)$ is reduced to a distribution over graphone sequences $p(q)$ which is modeled by an M-gram sequence model:

$$p(g, \varphi) = p(q_1^K) = \prod_{i=1}^{K+1} p(q_i | q_{i-1}, \ldots, q_{i-M+1})$$

- the segmentation into graphones may be non-unique, but we apply a maximum approximation
- the M-gram model can be estimated using maximum-likelihood (ML) EM training on an existing pronunciation dictionary
  - no prior letter to phoneme alignment is needed
  - the set of graphones Q is inferred automatically
  - parameters of the algorithm:

    L maximum number of letter/phonemes per graphone
    M span of M-gram model

Decoding

- **maximum approximation: we look for the most likely graphone sequence matching the given spelling and project it onto the phonemes:**

$$\varphi(g) = \varphi(\operatorname*{argmax}_{q \in Q^* | g(q) = g} p(q))$$

- **first-best search (n-best, if pronunciation variants are wanted)**

Training

- ML training can be performed using the EM algorithm (hidden variable: segmentation)
- identify model parameters with the M-gram probability $\vartheta_{q,h} \equiv p(q|h;\vartheta)$
- the re-estimation equations for the updated parameters ϑ' are:

$$
\begin{aligned}
e(q,h;\vartheta) &:= \sum_{i=1}^{N} \sum_{q \in S(g_i,\varphi_i)} p(q|g_i,\varphi_i;\vartheta)\, n_{q,h}(q) \\
&= \sum_{i=1}^{N} \sum_{q \in S(g_i,\varphi_i)} \frac{p(q;\vartheta)}{\sum\limits_{q' \in S(g_i,\varphi_i)} p(q';\vartheta)} n_{q,h}(q)
\end{aligned}
\tag{1}
$$

$$
\vartheta'_{q,h} = \frac{e(q,h;\vartheta)}{\sum_{q'} e(q',h;\vartheta)}
\tag{2}
$$

h	history $q_i \| q_{i-1},\ldots,q_{i-M+1}$
$n_{q,h}(q)$	number of occurrences of hq in q

Training (2): Evidence

The quantity $e(q, h; \vartheta)$

- is the expected number of occurrences of the graphone q with the history h in the training sample under the current set of parameters ϑ
- we call $e(q, h; \vartheta)$ the evidence for q in the context of h
- it can be calculated efficiently by a forward-backward procedure

Training (3): Smoothing

- Eq. 2 is an ML estimate, analogous to relative frequencies
- in language modeling this is known to perform poorly on unseen data

Idea: apply smoothing
sequence model: absolute discounting with interpolation

$$p(q|h) = \frac{\max\{e(q,h) - b_h, 0\}}{\sum_{q'} e(q',h)} + \lambda(h)\, p'(q|\bar{h}) \tag{3}$$

$b_h = b(\vert h\vert)$	set of M discounting parameters (to be optimized)
$\lambda(h)$	chosen to make the distribution sum up to one
$p'(q\vert\bar{h})$	generalized, lower order distribution ($M-1$-gram)

Zerogram back-off is included, which allows emergence of "new" graphones

Training (4): Optimization

- leaving-one-out: not clear how to apply this with fractional counts
- marginal constraints

Impose consistency constraint for all $\bar{h}$:

$$\sum_{h \in \bar{h}} p(q|h) \sum_{q'} e(q', h) = \sum_{h \in \bar{h}} e(q, h) \tag{4}$$

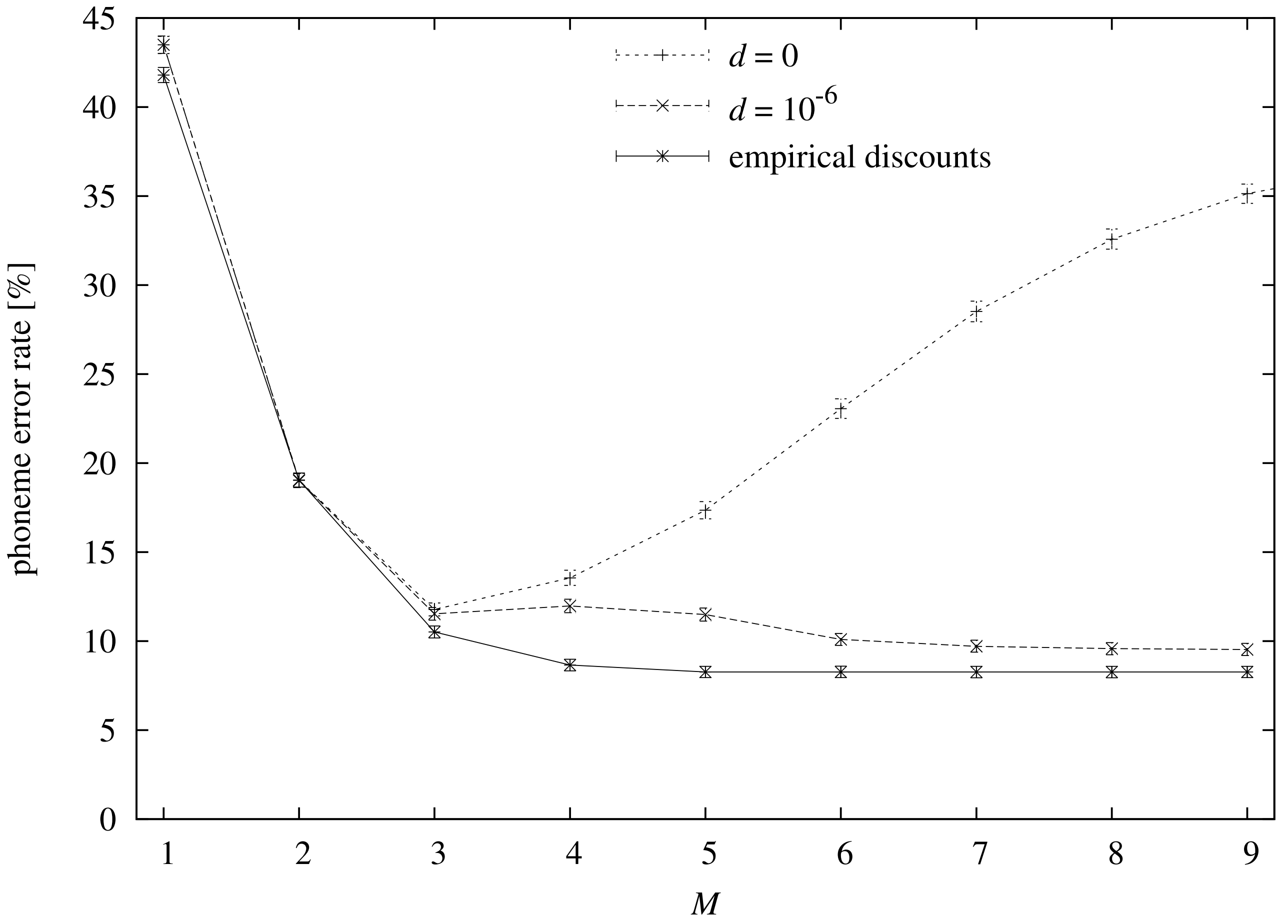
Substituting with Eq. 3 and solving for $p'(q|h)$ yields:

$$p'(q|\bar{h}) = \frac{\sum_{h \in \bar{h}} \min\{e(q, h), b_h\}}{\sum_{q'} \sum_{h \in \bar{h}} \min\{e(q', h), b_h\}} \tag{5}$$

- smoothing: "plugging in" the modified evidences in Eq. 3 is equivalent to smoothing the constraints in Eq. 4
- effectively a variable length history model: when the discount is larger than all evidence values, the distribution becomes identical to the ($M-1$)-gram model

Training (5): Effect of Smoothing

- **PER for pronunciation generation (NetTalk 15k, $L = 1$)**
- **smoothed and unsmoothed models as a function of context length M**



Training (6): Training recipe

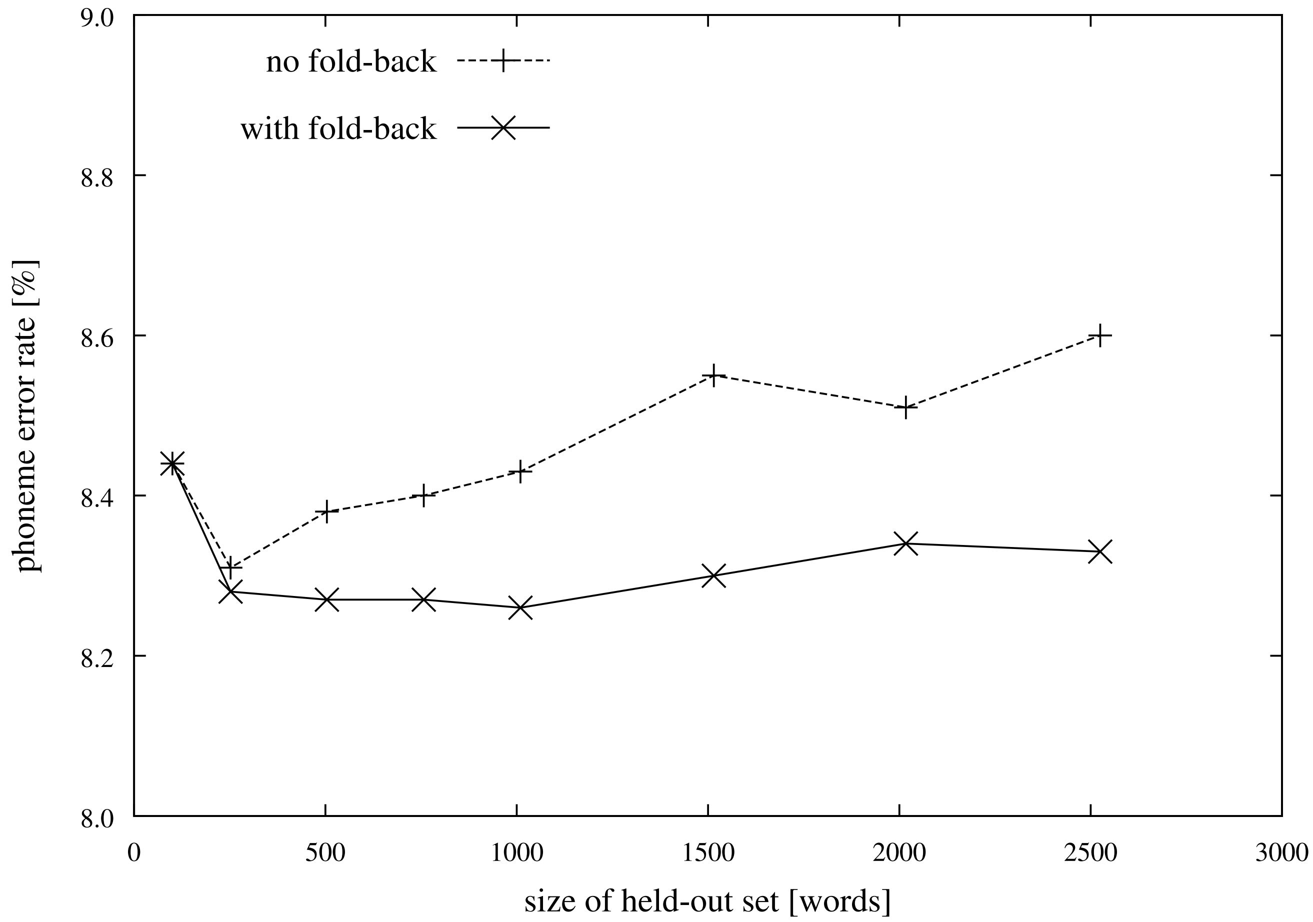
for M= 1 to $M_{\max}$:
  initialize M-gram model with (M-1)-gram model
    $p_M(q|h) = p_{M-1}(q|\bar{h})$
  initialize the additional discount parameter
    $b_M = b_{M-1}$
  repeat until $\mathcal{L}(\mathcal{O}_h)$ stops increasing:
    compute evidence according to (1)
    if $\mathcal{L}(\mathcal{O}_h)$ did not increase:
      adjust discount parameters $b_1, \ldots, b_{M-1}$ by direction set method
        $b = \operatorname{argmax}_{b'} \mathcal{L}(\mathcal{O}_h; b')$
    update model according to (3) and (5)

with held-out data $\mathcal{O}_h$
uniform initial distribution:

$$p_0(q) = \left[\sum_{l=0}^{L} \sum_{r=0}^{L} |G|^l |\Phi|^r \right]^{-1} \tag{6}$$

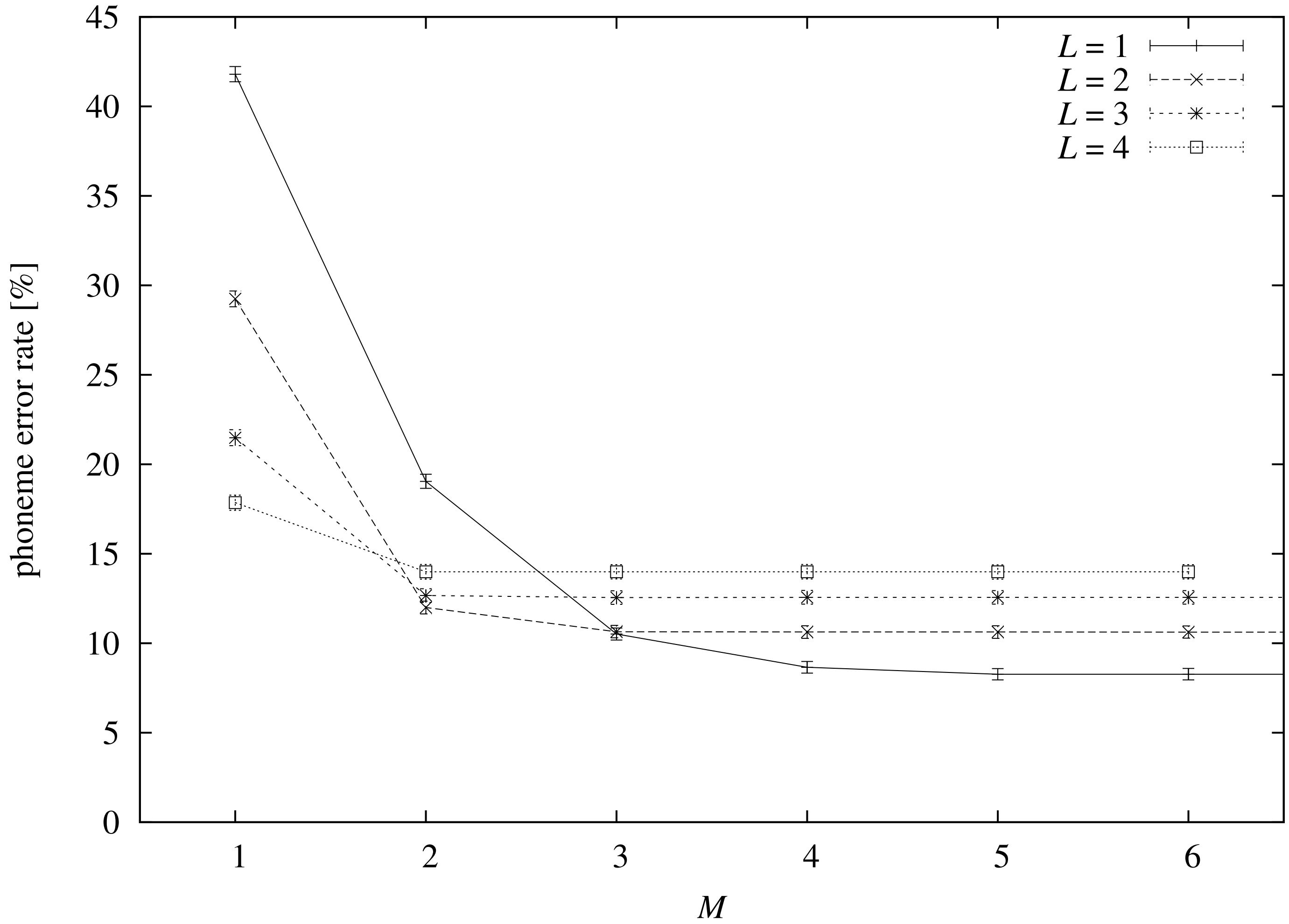
Training (7): Foldback

- common "trick": add the held-out data to the training data after parameters are optimized
- PER as a function of the size of the held-out set (NETtalk 15k, $L = 1$,$M = 6$)



Parameter Tuning

- M and L have to be tuned empirically
- here: NETtalk 15k database



Databases

	symbols		word length		prons/	number of words		
	$\|G\|$	$\|\Phi\|$	$\|g\|$	$\|\varphi\|$	words	train	test	held
British English								
Celex	26	53	8.4	7.1	1	39,995	15,000	5,000
OALD	26	82	8.2	6.9	1.008	56,961	6,377	–
US English								
NETtalk 15k	26	50	7.3	6.2	1.010	14,851	4,951	–
CMUdict	27	39	7.5	6.3	1.062	106,837	12,000	–
Pronlex	30	41	7.4	6.9	1.094	83,182	4,800	2,400
German								
LexDb	30	46	10.4	9.0	1	40,000	15,000	5,000
French								
Brulex	40	39	8.5	6.7	1	24,726	2,747	–

Results

data set		author	PER [%]		WER [%]	
Celex	=	[Vozila & Adams+ 03]	3.68		17.13	
	=	this method	2.50	± 0.11	11.42	± 0.43
OALD		[Pagel & Lenzo+ 98]	6.03		21.87	
	=	this work	3.54	± 0.19	17.49	± 0.78
NETtalk 15k		[Jiang & Hon+ 97]	8.1		34.2	
	=	this work	8.26	± 0.32	33.67	± 1.10
CMUdict	=	[Chen 03]	5.9		24.7	
	=	this work	5.88	± 0.18	24.53	± 0.65
Pronlex	=	[Chen 03]	7.15		27.3	
	=	this work	6.78	± 0.31	27.33	± 1.04
LexDb (German)	=	this work	0.28	± 0.03	1.75	± 0.18
Brulex (French)	=	this work	1.18	± 0.05	6.25	± 0.24

(± indicates 90% confidence interval)

Conclusion

Grapheme-to-Phoneme Conversion
is an indispensable component of (almost) any practical large vocabulary speech recognition system

Joint-Sequence Models
make highly accurate G2P converters:

- data driven and language independent (learn only from examples)
- best results obtained with only trivial graphones (no chunking)
- long-range M-gram models are needed (“remember” everything)

are versatile:

- n-best generation
- confidence estimation
- phoneme set conversion

Training Using Sequitur G2P (1)

- the Sequitur G2P toolkit consists of a number of tools
- we will use the `g2p.py` script to train and apply the g2p models
- other tools may be used to estimate LMs or build (basic) open vocabulary models (e.g. `makeOvModel.py`)
- important options:
  - `-train=FILE`: training dictionary
  - `-devel=FILE`: development dictionary
  - `-size-constraint l1,l2,r1,r2`: context length left/right
  - `-self-test`: evaluate the performance on a development set
  - `-write-model=FILE`: write the current model to FILE

Training Using Sequitur G2P (2)

- `g2p/g2p_model_training.sh`
- **(pre-trained g2p models available online)**
- **measure: lowest symbol error rate**
- **example of training log file:** `g2p/log/voxforge-m7.log`

Results for Voxforge ($L = 1$):

M	1	2	3	4	5	6	7	8	9
PER[%]	46.32	20.35	10.83	7.63	6.68	6.50	6.47	6.48	6.48
WER[%]	99.13	70.76	43.55	31.13	27.60	26.87	26.62	26.67	26.67

Building of Recognition Lexica

`g2p/build_recognition_lexica.sh`

- next steps: convert vocabulary files, e.g. `data/lm/vocab.3k`, to pronunciation dictionaries:

1.) match the words to the base pronunciation dictionary

2.) for all words not in the base dictionary (“fail” files) apply g2p model

3.) discard words which could not be processed by g2p and build pronunciation dictionary

4.) convert pronunciation dictionary to R*ASR* recognition lexicon

Why Does Sequitur G2P Fail Sometimes?

- **foreign names :**
  - **abbé**
  - **tête**
  - **cañon**
- **garbage :**
  - **&c**
  - **=**
- **all these “special” characters have not been seen in training**
- **circumvention: map these characters to close ones in the target language in a preprocessing step**

Outline

- Introduction and Goals
- Acoustic Model Training using R*ASR*
- Vocabulary Selection and Pronunciation Generation
- Language Model Training using SRI LM
- Decoding using R*ASR* and Evaluation
- Open Vocabulary Recognition and Evaluation
- Outlook

Language Modeling: Performance Measure

we will use the perplexity PP:

$$PP := \left[Pr(w_1^L)\right]^{-1/L}$$

- **interpretation: number of choices per word position**
- **lower perplexities are better**
- **typical perplexities for an English text:** $40 - 1000$
- **depends on**
  - **test corpus** w_1^L
  - **language model** $p(w|h)$
  - **vocabulary**

language model (LM): prior probability in Bayes decision rule for x_1^T:

$$\max_{w_1^L}\{Pr(w_1^L)\cdot Pr(x_1^T|w_1^L)\}$$

- **rewrite using "chain rule" and Markov assumption of order n**

$$p(w_1^L) = \prod_{i=1}^{L} p(w_i|w_{i-n+1}^{i-1})$$

ML estimate (relative frequency):

$$p(w|h) = \frac{N(h,w)}{\sum_{w'} N(h,w')} = \frac{N(h,w)}{N(h,\cdot)}$$

$h := w_{i-n+1}^{i-1}$ **(history)**

$N(h,w) := \sum_{(h',w')} \delta(h,h')\delta(w,w')$ **(n-gram count)**

Language Modeling: Theory (2)

Examples (taken from [Chen & Goodman 98])
Training data: 3 sentences

`JOHN READ MOBY DICK, MARY READ A DIFFERENT BOOK, SHE READ A BOOK BY CHER`

Bigram ML estimates for p(`JOHN READ A BOOK`):

$$p(\texttt{JOHN}|\langle \texttt{s} \rangle) = \frac{N(\langle \texttt{s} \rangle, \texttt{JOHN})}{N(\langle \texttt{s} \rangle, \cdot)} = \frac{1}{3}$$

$$p(\texttt{READ}|\texttt{JOHN}) = \frac{N(\texttt{JOHN}, \texttt{READ})}{N(\texttt{JOHN}, \cdot)} = \frac{1}{1}$$

$$p(\texttt{A}|\texttt{READ}) = \frac{N(\texttt{READ}, \texttt{A})}{N(\texttt{READ}, \cdot)} = \frac{2}{3}$$

$$p(\texttt{BOOK}|\texttt{A}) = \frac{N(\texttt{A}, \texttt{BOOK})}{N(\texttt{A}, \cdot)} = \frac{1}{2}$$

$$p(\langle /\texttt{s} \rangle|\texttt{BOOK}) = \frac{N(\texttt{BOOK}, \langle /\texttt{s} \rangle)}{N(\texttt{BOOK}, \cdot)} = \frac{1}{2}$$

Result:

$$\begin{aligned} p(\texttt{JOHN READ A BOOK}) &= p(\texttt{JOHN}|\langle \texttt{s} \rangle)\, p(\texttt{READ}|\texttt{JOHN})\, p(\texttt{A}|\texttt{READ})\, p(\texttt{BOOK}|\texttt{A})\, p(\langle /\texttt{s} \rangle|\texttt{BOOK}) \\ &= \frac{1}{3} \cdot \frac{1}{1} \cdot \frac{2}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} \\ &\approx 0.06 \end{aligned}$$

Language Modeling: Smoothing (1)

Problem: consider `CHER READ A BOOK`:

$$p(\texttt{READ}|\texttt{CHER}) = \frac{N(\texttt{CHER}, \texttt{READ})}{N(\texttt{CHER}, \cdot)} = \frac{0}{1}$$

- $p(\texttt{CHER READ A BOOK}) = 0$
- **underestimate of the probability**
- **idea: smoothing (adjusting of ML estimate)**

Language Modeling: Smoothing (2)

add-one-smoothing: pretend each bigram occurs once more than it actually does ($|V|$= vocabulary size)

$$p(w|h) = \frac{1 + N(h,w)}{\sum_{w'}(1 + N(h,w'))} = \frac{1 + N(h,w)}{|V| + N(h,\cdot)}$$

$$\begin{aligned} p(\texttt{JOHN READ A BOOK}) &= p(\texttt{JOHN}|\langle\texttt{s}\rangle)\, p(\texttt{READ}|\texttt{JOHN})\, p(\texttt{A}|\texttt{READ})\, p(\texttt{BOOK}|\texttt{A})\, p(\langle/\texttt{s}\rangle|\texttt{BOOK}) \\ &= \frac{2}{14} \cdot \frac{2}{12} \cdot \frac{3}{14} \cdot \frac{2}{13} \cdot \frac{2}{13} \\ &\approx 0.0001 \end{aligned}$$

$$\begin{aligned} p(\texttt{CHER READ A BOOK}) &= p(\texttt{CHER}|\langle\texttt{s}\rangle)\, p(\texttt{READ}|\texttt{CHER})\, p(\texttt{A}|\texttt{READ})\, p(\texttt{BOOK}|\texttt{A})\, p(\langle/\texttt{s}\rangle|\texttt{BOOK}) \\ &= \frac{1}{14} \cdot \frac{1}{12} \cdot \frac{3}{14} \cdot \frac{2}{13} \cdot \frac{2}{13} \\ &\approx 0.00003 \end{aligned}$$

- **add-one-smoothing overestimates unseen events**
- **more sophisticated smoothing techniques described in the literature**

Language Modeling: Smoothing (3)

Witten-Bell smoothing: nth order model is defined recursively:

$$p_{\mathrm{WB}}(w|h) = \lambda_h p_{\mathrm{ML}}(w|h) + (1-\lambda_h)p_{\mathrm{WB}}(w|\bar{h})$$

$h := w_{i-n+1}^{i-1}$
$\bar{h} := w_{i-n+2}^{i-1}$ **(shortened history)**

with $1-\lambda_h = \frac{N^{1+}(h,\cdot)}{N^{1+}(h,\cdot)+N(h,\cdot)}$ **(normalization constraint), we get:**

$$p_{\mathrm{WB}}(w|h) = \frac{N(h,w) + N^{1+}(h,\cdot)\, p_{\mathrm{WB}}(w|\bar{h})}{N(h,\cdot) + N^{1+}(h,\cdot)}$$

$N(h,w) := \sum_{(h',w')} \delta(h,h')\delta(w,w')$ **(n-gram count)**
$N^{1+}(h,\cdot) := |\{w : N(h,w) > 0\}|$ **(right diversity count)**

[Witten & Bell 91]

interpolated modified Kneser-Ney smoothing:

- **three discount parameters**
- **applied to n-grams with one, two, and three or more observations:**

$$p_{\mathrm{KN}}(w|h) = \frac{N(h,w) - D(N(h,w))}{N(h,\cdot)} + \gamma_h p_{\mathrm{KN}}(w|\bar{h})$$

with

$$D(c) = \begin{cases} 0 & \text{if } c = 0 \\ D_1 & \text{if } c = 1 \\ D_2 & \text{if } c = 2 \\ D_{3+} & \text{if } c \geq 3 \end{cases}$$

Language Modeling: Smoothing (5)

how to estimate lower-order estimate $p_{\mathrm{KN}}(w|\bar{h})$?

- **not just recursively**
- **impose consistency constraints:**

$$p(w|\bar{h}) = \sum_{h \in \bar{h}} p(h, w|\bar{h})$$

- **result: for lower order n-grams, replace n-gram counts $N(\bar{h}, w)$ with modified counts:**

$$p_{\mathrm{KN}}(w|\bar{h}) = \frac{N_+(\cdot\bar{h}, w) - D(N_+(\cdot\bar{h}, w))}{N_+(\cdot\bar{h}, \cdot)} + \gamma_{\bar{h}} p_{\mathrm{KN}}(w|\bar{\bar{h}})$$

where

$N_+(\cdot\bar{h}, w) := |\{h \in \bar{h} : N(h, w) > 0\}|$ **(left diversity count)**

[Kneser & Ney 95]

Language Modeling: Smoothing (6)

to make the distribution sum up to 1:

$$\gamma_h = \frac{D_1 N^1(h,\cdot) + D_2 N^2(h,\cdot) + D_{3+} N^{3+}(h,\cdot)}{N(h,\cdot)}$$

discounts are estimated from the training data count-counts ($n_1 \ldots n_4$):

$$\begin{aligned} n_r &:= \sum_h N^r(h,\cdot) \\ &= \sum_{hw:N(h,w)=r} 1 \end{aligned}$$

$$\begin{aligned} Y &= \frac{n_1}{n_1 + 2n_2} \\ D_1 &= 1 - 2Y\frac{n_2}{n_1} \\ D_2 &= 2 - 3Y\frac{n_3}{n_2} \\ D_{3+} &= 3 - 4Y\frac{n_4}{n_3} \end{aligned}$$

Why These Two Smoothing Techniques?

- experimentally verified by [Chen & Goodman 98]:
  modified Kneser-Ney discounting is the method of choice
- best performing method w.r.t. perplexity
- disadvantage: does not work if one of the four count of counts is zero
- possible solutions:
  - if singletons/doubletons exist in the corpus: use unmodified Kneser-Ney smoothing
    (only one discount is necessary)
  - or: optimize discounts on held-out data (not possible with SRI Toolkit)
  - easy solution: use smoothing method not relying on discounts
- our solution: for those orders where problems occur, use Witten-Bell smoothing instead

Language Modeling: Conclusion

Why?

- together with the acoustic model, the LM builds the basis of every ASR system
- usually, it gives probabilities for short sequences of words (n-grams)
- and thus puts context information (syntax, semantics) into the ASR decoding process
- it is trained using large amounts of natural language text data

Smoothing

- many words do not occur in the training data
- to assign probabilities to these words, smoothing techniques are applied (like the absolute discounting in Sequitur G2P)
- method of choice: modified Kneser-Ney smoothing

SRI LM: Language Model Training

- we are using the SRI LM modelling toolkit, especially the tools `ngram-count` and `ngram`
- idea:
  - build LMs from each of the already normalized text sources:
    - Project Gutenberg background texts
    - transcripts of audio data
  - using vocabularies, `voxforgeDict.{3k,6k,12k}.vocab`
  - using modified Kneser-Ney smoothing
  - and the unknown word token `[UNKNOWN]`
  - build LMs for n-gram orders 2, 3 and 4
  - use perplexity on the test set as measurement to assess the quality of the model

[Stolcke 02]

SRI LM: Language Model Training (2)

lm/lm_training.sh

```
$NGRAMCOUNT                 \
-text $part                 \
-order $order               \
-kndiscount1                \
-kndiscount2                \
-kndiscount3                \
-kndiscount4                \
-interpolate                \
-kn1 $target.kn1            \
-kn2 $target.kn2            \
-kn3 $target.kn3            \
-kn4 $target.kn4
```

```
$NGRAMCOUNT                 \
-text $part                 \
-vocab $vocab               \
-unk                        \
-map-unk '[UNKNOWN]'        \
-order $order               \
-kndiscount1                \
-kndiscount2                \
-kndiscount3                \
-kndiscount4                \
-interpolate                \
-kn1 $target.kn1            \
-kn2 $target.kn2            \
-kn3 $target.kn3            \
-kn4 $target.kn4            \
-lm $target.lm
```

- **first call to `ngram-count`: calculate discounts per order (unlimited vocabulary)**
- **second call: use the discounts to estimate LM (limited vocabulary)**

Effect of n-gram order on Perplexity

perplexity results for the three vocabulary sizes trained on

- `data/lm/background.text` **(BG)**
- `data/lm/train.uniq.text` **(AT)**

using only modified Kneser-Ney smoothing

vocab. size	corpus	n-gram order 1	2	3	4
3k	BG	183.79	72.45	58.61	56.91
	AT	159.15	37.35	35.88	35.59
6k	BG	289.56	105.10	85.03	83.18
	AT	240.45	42.55	41.60	41.46
12k	BG	457.72	157.83	127.78	125.01
	AT	354.69	46.78	46.60	46.59

- **only small improvements from 3- to 4-gram**
- **4-gram LMs give best performance**

Interpolation of LMs

- idea:
  - interpolate the LMs from both text sources for each order
  - optimize the interpolation weight w.r.t. perplexity on the test set
  - for 2 LMs: grid search suffices
  - $>$ 2 LMs: more sophisticated methods (e.g. downhill-simplex)
  - choose final LM

SRI LM: Interpolation of LMs

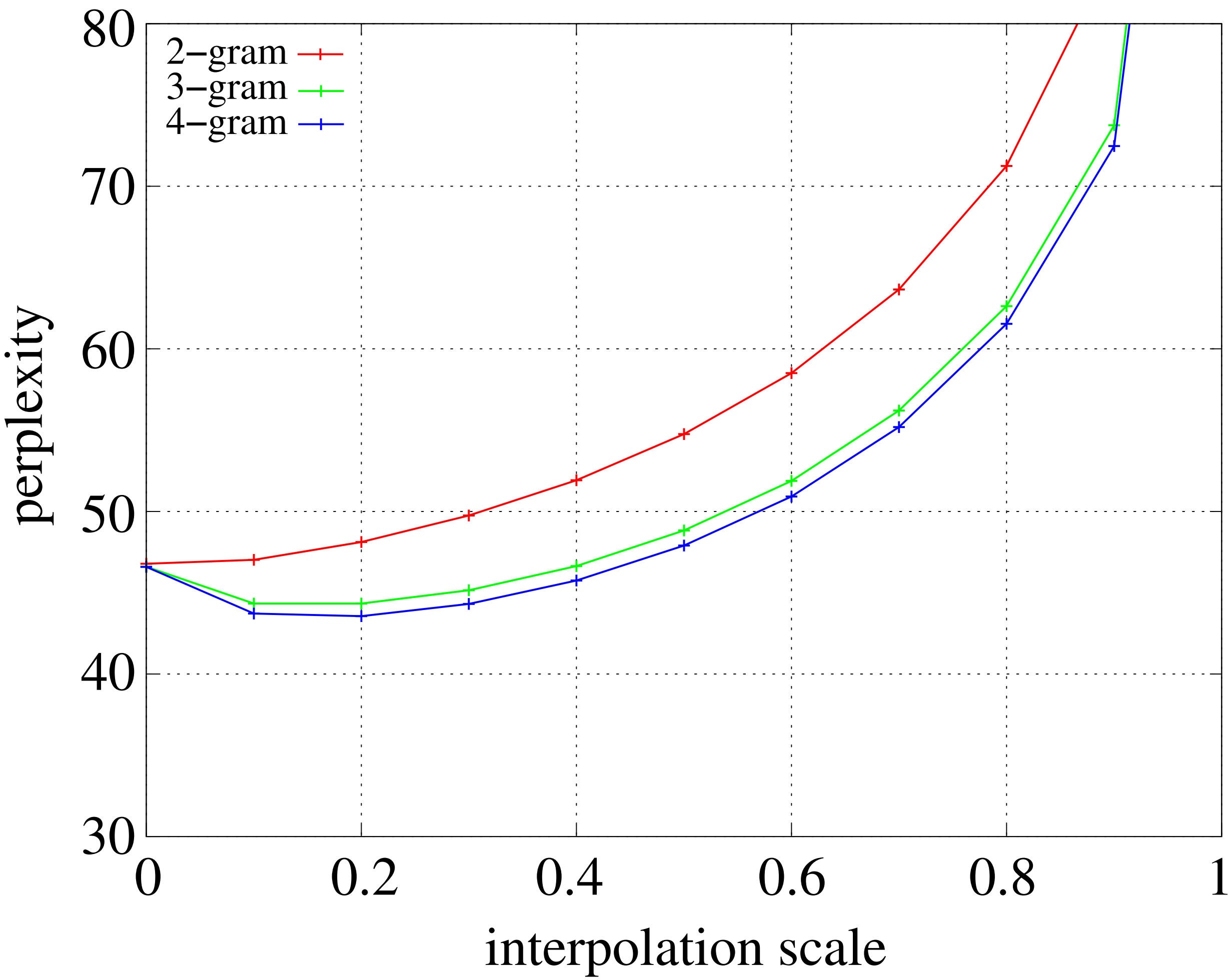
lm/lm_training.sh

```
for order in 2 3 4 ; do
    echo "interpolated    order $order"
    for lambda in 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 ; do
        echo "lambda = $lambda"
        target=$targetprefix-M$order-L$lambda.sri
            $NGRAM \
            -vocab $vocab           \
            -unk                    \
            -map-unk '[UNKNOWN]'    \
            -order $order           \
            -renorm                 \
            -lm $input1-M$order.sri.lm.gz \
            -mix-lm $input2-M$order.sri.lm.gz \
            -lambda $lambda         \
            -ppl test.uniq.text    \
            -write-lm $target.lm

            gzip $target.lm
    done
done
```

- **interpolation of 2 LMs using grid search**
- **afterwards: delete all but the best performing LM per order**

Effect of Interpolation on Perplexity

here: 12k vocabulary

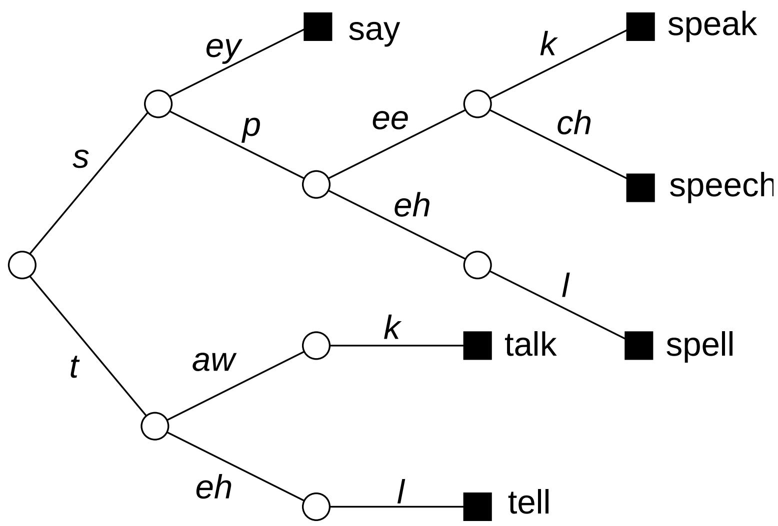


Outline

- Introduction and Goals
- Acoustic Model Training using R*ASR*
- Vocabulary Selection and Pronunciation Generation
- Language Model Training using SRI LM
- Decoding using R*ASR* and Evaluation
- Open Vocabulary Recognition and Evaluation
- Outlook

RASR: Decoder

- search: history conditioned lexical tree search
- one-pass dynamic programming algorithm
- uses pre-compiled lexical prefix tree (pronunciation dictionary)



- search space structure:
  - word identity only known at leaf nodes
  - LM probability can only be applied at word ends
  - introduce "tree copies" for word histories
    e.g. preceding 2 words are required for 3-gram LM

[Ney & Haeb-Umbach^{+} 92, Ney & Ortmanns 00]

RASR: Pruning

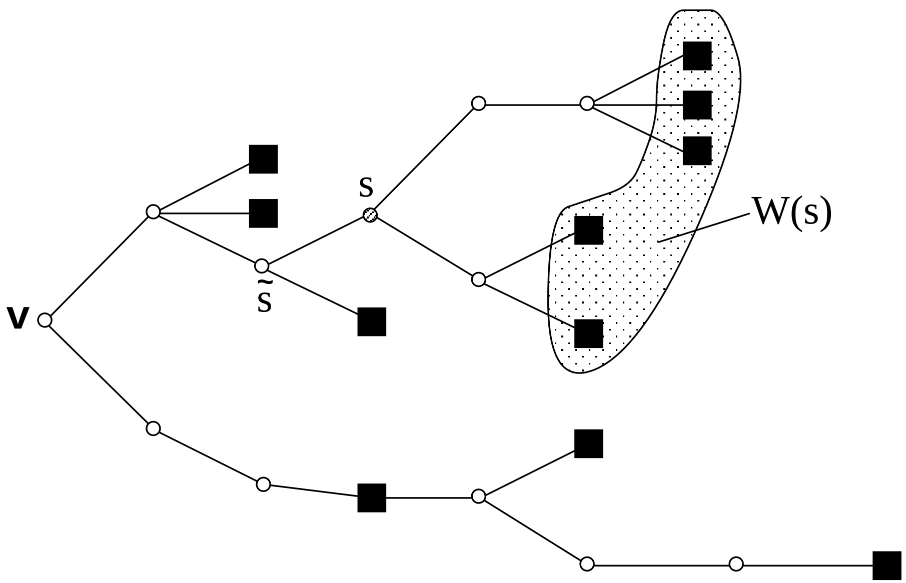
beam search: keep only most promising hypotheses

- **acoustic pruning**
  - determine score of best hypothesis at each time t: $Q_{AC}(t)$
  - beam pruning: eliminate hypothesis if its score is higher than $f_{AC} \cdot Q_{AC}(t)$
  - `acoustic-pruning`: beam size ($\log(f_A C)$) for state hypotheses
  - `acoustic-pruning-limit`: limit absolute number of state hypotheses

- **language model pruning**
  - prune word end hypotheses
  - `lm-pruning`: beam size for word end hypotheses
  - `lm-pruning-limit`: limit absolute number of word end hypotheses

R*ASR*: LM Look-ahead

- idea: incorporate language model as early as possible in the search process
- from a certain state only a small subset of word ends can be reached
- use the best possible score to refine acoustic pruning
- configuration (`lm-lookahead`)
  - `history-limit`: length of history considered for LM look-ahead
  - `tree-cutoff`: maximum depth of state tree covered by LM look-ahead



[Ortmanns & Ney 00]

RASR: Model Combination

$$\operatorname*{argmax}_{w_1^N} \left\{ \sum_{n=1}^{N} \log \overbrace{p(w_n|w_1^{n-1})^{\alpha}}^{\text{language model}} + \max_{s_1^T} \sum_{t=1}^{T} \left[\log \underbrace{p(s_t|s_{t-1}, w_1^N)^{\beta}}_{\text{transition model}} + \log \underbrace{p(x_t|s_t, w_1^N)^{\gamma}}_{\text{acoustic model}} \right] \right\}$$

- α **language model scale:** `lm.scale`
- β **transition probability scale:** `acoustic-model.tdp.scale`
- γ **acoustic model scale:** `acoustic-model.mixture-set.scale`

- **pronuncation scale: weight the pronuncation scores (if existing)** `model-combination.pronunciation-scale`

- **usually: tune only LM scale**

Evaluation of Recognition Results (1)

result: log file `recognition/log/recognition.log`

```
&lt;segment name="1" full-name="CMU_ARCTIC/bdl_b0474/1" track="0" start="0" end="inf">
  <speaker name="bdl" gender="unknown"/>
  <orth source="reference">
    he was manifestly distressed by my coming
  </orth>
  <traceback>
    t=0       s=0         #|#
    t=1       s=29.7992   [SILENCE]           /si/    #|#
    t=15      s=597.201   he                  /hh iy/    #|#
    t=31      s=1135.63   was                 /w ax z/    #|#
    t=87      s=3288.43   manifest            /m ae n ax f eh s t/    #|#
    t=103     s=4164.02   laid                /l ey d/    #|#
    t=151     s=6167.88   distressed          /d ix s t r eh s t/    #|#
    t=154     s=6301.97   [SILENCE]           /si/    #|#
    t=166     s=6745.41   by                  /b ay/    #|#
    t=186     s=7543.7    my                  /m ay/    #|#
    t=227     s=9115.72   coming              /k ah m ix ng/    #|#
    t=232     s=9343.76   [SILENCE]           /si/    #|#
    t=233     s=9409.29   #|#
  </traceback>
  <orth source="recognized">
    [SILENCE] he was manifest laid distressed [SILENCE] by my coming [SILENCE]
  </orth>
```

Evaluation of Recognition Results (2)

- evaluate recognition results: `recognition/score.sh`
- create CTM file from recognition log file using `analog`

`recognition/log/recognition.ctm`

```
clb_b0020 1 2.380 0.230 more
clb_b0020 1 2.610 0.030 @
clb_b0020 1 2.640 0.090 to
clb_b0020 1 2.730 0.640 continue
clb_b0020 1 3.370 0.230 @
bdl_b0474 1 0.000 0.010 @
bdl_b0474 1 0.010 0.140 he
bdl_b0474 1 0.150 0.160 was
bdl_b0474 1 0.310 0.560 manifest
bdl_b0474 1 0.870 0.160 laid
bdl_b0474 1 1.030 0.480 distressed
bdl_b0474 1 1.510 0.030 @
bdl_b0474 1 1.540 0.120 by
bdl_b0474 1 1.660 0.200 my
bdl_b0474 1 1.860 0.410 coming
bdl_b0474 1 2.270 0.050 @
slt_a0584 1 0.000 0.170 @
slt_a0584 1 0.170 0.100 i
slt_a0584 1 0.270 0.200 have
```

`xml/test_bdl_slt_clb_rms.stm`

```
clb_b0020 1 clb 0 inf he made no reply as he waited
                      for whittemore to continue
bdl_b0474 1 bdl 0 inf he was manifestly distressed
                      by my coming
slt_a0584 1 slt 0 inf i have long noted your thirst
                      unquenchable
```

Evaluation of Recognition Results (3)

run NIST sclite to evaluate results

recognition/log/recognition.dtl

```
WORD RECOGNITION PERFORMANCE

Percent Total Error        =   14.0%   ( 577)

Percent Correct            =   90.6%   (3722)

Percent Substitution       =    8.8%   ( 360)
Percent Deletions          =    0.6%   (  26)
Percent Insertions         =    4.6%   ( 191)
```

recognition/log/recognition.pra

```
File: bdl_b0474
Scores: (#C #S #D #I) 6 1 0 1
REF:  he was ******** MANIFESTLY distressed by my coming
HYP:  he was MANIFEST LAID       distressed by my coming
Eval:        I        S
```

recognition/log/recognition.sys

```
|   SPKR        |   # Snt        # Wrd   |   Corr         Sub        Del         Ins         Err
|---------------+------------------------+--------------------------------------------------------
|   bdl         |    114          1027   |   90.0         9.2        0.9         4.7        14.7
|---------------+------------------------+--------------------------------------------------------
|   clb         |    114          1027   |   90.9         8.7        0.4         4.2        13.2
|---------------+------------------------+--------------------------------------------------------
|   rms         |    114          1027   |   91.4         8.3        0.3         5.2        13.7
```

Recognition Results: Acoustic Models

	#densities	WER	del.	ins.	sub
monophone models	15,603	16.5	2.9	2.7	10.9
triphone models	62,370	14.0	0.6	4.6	8.8

- 4-gram LM
- very low amount of training data
- results not representative
- not well tuned parameters (left as an exercise)

Recognition Results: Language Models

		LM order				
vocabulary	OOV rate	0	1	2	3	4
3k	17.42	47.2	38.4	31.1	30.0	30.1
6k	12.40	41.0	30.4	22.8	21.4	21.5
12k	7.50	39.0	22.6	15.5	14.1	14.0

- high OOV rates
- OOV rate is lower bound for WER
- rule of thumb: OOV rate $\times 2 \approx$ WER

Recognition Results: Examples

Vocab. Size	File	Date	Status	Result	
12k	closedvocab_rms_a.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_awb_a.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_clb_a.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_clb_b.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_awb_b.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_rms_b.wav	2011-02-03 11:26:49		Open	View
12k	baseline_awb_b.wav	2011-02-02 19:16:09		Open	View
12k	baseline_clb_b.wav	2011-02-02 19:16:09		Open	View
12k	baseline_rms_b.wav	2011-02-02 19:16:09		Open	View
12k	baseline_clb_a.wav	2011-02-02 19:16:09		Open	View
12k	baseline_awb_a.wav	2011-02-02 19:16:09		Open	View
12k	baseline_rms_a.wav	2011-02-02 19:16:09		Open	View

- **more insertion errors than deletion errors:**
  OOV words are often recognized as several short words
- **optimized HMM skip transition penalty is low:**
  short substituted words used as “word fragments”

```
REF:  your face was the ***** ** PERSONIFICATION of ********* DUPLICITY
HYP:  your face was the PURSE ON VACATION        of DUPLESSIS SEE
Eval:                   I     I  S                  I         S
```

Outline

- Introduction and Goals
- Acoustic Model Training using R*ASR*
- Vocabulary Selection and Pronunciation Generation
- Language Model Training using SRI LM
- Decoding using R*ASR* and Evaluation
- Open Vocabulary Recognition and Evaluation
- Outlook

Extension: Open Vocabulary Recognition (1)

- **LVCSR systems operate with a fixed large but finite vocabulary (typically 10k – 100k [Arabic: $\gg$100k] word forms)**
- **fixed vocabulary suitable for e.g. dictation in a fixed domain, but less suitable for open vocabulary settings like broadcast news:**
  - **number of different words does not appear to be finite**
  - **important content words change over time**
- **fixed vocabulary implies out-of-vocabulary (OOV) words, which:**
  - **are never recognized, and are substituted by in-vocabulary word[s]**
  - **lead to misrecognition of neighboring words**
  - **lead to errors that cannot be recovered by later processing stages (e.g. translation, understanding, document retrieval)**
  - **often are content words**

goal: systems that can

- **detect *and* recognize OOVs**
- **handle any spoken word without help from the user**

Extension: Open Vocabulary Recognition (2)

idea:

- add word fragments to recognition lexicon

aims:

- allow for recognition of more or less arbitrary grapheme sequences
- minimize effect on in-vocabulary words

language model incl. OOVs: flat hybrid approach

- generalization of LM using standard techniques:
  replace OOVs in (usual) LM training corpus by word fragments
  - **hybrid**: contains mixed M-grams composed of **words** and **fragments**
  - **flat**: a single model for in-vocabulary and OOV words, fixed context length

search with open vocabulary:

- conventional decoder
- unified set of recognition units with (sub-lexical) language model

Extension: Open Vocabulary Recognition (3)

WSJ Dictation Task (NAB 93/94 Dev HUB 1) [Bisani & Ney 05]

- baseline vocabularies: 5k (11% OOV), 20k (2.6% OOV), 64k (0.5% OOV)
- relative improvements in WER of up to 30% for high OOV rates (5k)
- no deterioration when including OOV model at large vocabularies/low OOV
- average errors per OOV word decrease by about 0.5 at all vocabularies
- about 30% of the OOVs are recognized correctly
- effect on both OOV and in-vocabulary errors

GALE Arabic Broadcast News [El-Desoky Mousa & Schlüter^{+} 10]

baseline lexicon number of			open vocabulary additional			WER [%]		
						OOV	baseline	open
words	pron.	PP	fragm.	pron.	PP	rate	lexicon	vocabulary
64k	125k	674	8.7k	13k	854	5.2	22.6	21.2
126k	232k	736	8.7k	13k	962	2.9	20.9	20.4
256k	423k	793	8.6k	12k	989	1.3	20.5	20.1

conclusion: out-of-vocabulary words can be both detected and recognized

Extension: Open Vocabulary Recognition (4)

recipe:

- fix a certain base vocabulary (full words)
- extract all OOVs w.r.t. the base vocabulary from the LM training texts
- segment the OOVs into fragments
- replace the OOVs by the fragments in the LM training texts + test text
- generate count statistics and select a hybrid-vocabulary
- generate pronunciations for the selected fragments and full words
- build a recognition lexicon using fragments and full words
- build an LM using the modified LM training texts
- do a recognition pass
- merge the fragments to form words prior to scoring

Extension: Open Vocabulary Recognition (5)

in this tutorial: two different ways of fragmenting words

- **statistical segmentation based on morphemes**
- **statistical segmentation based on graphones**

we will start with the morpheme-based approach

- **morpheme: smallest conceptual meaningful component of a word, or other linguistic unit, that has semantic meaning**
- **example: un break able**

[Bisani & Ney 05, El-Desoky Mousa & Schlüter^{+} 10, El-Desoky Mousa & Basha Shaik^{+} 10, Basha Shaik & El-Desoky Mousa^{+} 11]

OOV Extraction

`ov-lm/oov_extraction.sh`

- extract all words from the training texts and sort them w.r.t. their frequency
- extract full word vocabulary
- all other words: take as OOVs (but not those seen less than n times, here: 4)
- result: a number of OOV words w.r.t. `voxforgeDict.{3k,6k,12k}.vocab`
- these words will be used to train the hybrid models

Segmentation

ov-lm/apply_segmenter.sh

- we apply the open-source tool `morfessor.pl`
- data-driven approach considering graphemes only
- morfessor generates "linguistically motivated" segmentations:

```
bonaparte bona+ par+ te
bonaventure bona+ venture
bonbright bon+ bright
bond-street bond-+ street
boneset bone+ set
bonfire bon+ fire
bonfires bon+ fires
bong bong
```

- problematic: word fragments are often real words
- this can lead to confusion for the models

Generate Hybrid LM Text Sources and Vocabulary

`ov-lm/generate_hybrid_lm_text_sources_and_vocabulary.sh`

- basically the same procedure as with the closed-vocabulary LM
- `scripts/enrichLmTextWithFragments.py` is used to apply the segmentation to the OOVs of the LM training texts
- both newly generated training source files are merged into `background+train-hybrid.${VOCABSIZE}.text`
- a sorted count file with the most frequents counts at the top is generated
- the top `${NEWSIZE}` occurring tokens are selected and saved into `hybrid.vocab.${NEWSIZE}`, e.g. `data/ov-lm/hybrid.vocab.6k`
- `${NEWSIZE}`: 6k, 12k, 24k entries respectively to account for word fragments
- rationale: add to the 3k, 6k, 12k full words roughly the same number of word fragments

Generate Hybrid Recognition Lexicon

`ov-g2p/generate_hybrid_recognition_lexica.sh`

- **basically the same recipe as for the closed-vocabulary lexicon**
- **the same g2p model is used as for the closed-vocabulary lexicon**
- **input vocabulary: the hybrid vocabulary we just created**
- **for word fragments, the fragment marker + is removed prior applying the g2p model and restored afterwards**

`ov-lm/ov_lm_training.sh`

- **training procedure of hybrid LMs: basically the same as for regular LMs**
- **differences:**
  - **higher n-gram orders due to the mix of fragments and words (context information)**
  - **Kneser-Ney discounting fails for some n-gram orders due to lack of singletons/doubletons/. . . (we use Witten-Bell discounting instead)**
- **after LM interpolation (again grid search), choose hybrid LM with lowest perplexity on `data/ov-lm/test.uniq-hybrid.${NEWSIZE}.text`**
- **next step: a recognition pass using matching hybrid LM and lexicon**

Effect of n-gram order on Perplexity

Perplexity-Results for three vocabulary sizes

vocab. size	n-gram order 2	3	4	5	6	7
6k	43.18	39.98	38.35	37.31	37.14	37.11
12k	44.48	42.28	41.42	40.43	40.27	40.24
24k	46.43	44.87	44.10	43.17	43.00	42.97

- only small improvements from 2- to 7-gram
- 7-gram LMs give best performance

Joining of Fragments and Scoring

- fragments need to be merged for scoring

```
rms_a0444 1 1.490 0.210 in
rms_a0444 1 1.700 0.200 al+
rms_a0444 1 1.900 0.330 tru+
rms_a0444 1 2.230 0.460 ism
rms_a0444 1 2.690 0.120 @
```

- join consecutive fragments in the generated NIST CTM file

```
rms_a0444 1 1.490 0.210 in
rms_a0444 1 1.700 0.990 altruism
rms_a0444 1 2.690 0.120 @
```

- `recognition/score.sh`
  - uses `scripts/joinfragments.py`
  - joins the fragments in the CTM file before executing sclite

Recognition Results: Open Vocabulary

morpheme-based segmentation [mb]:

	vocabulary		WER for n-gram order			
model	words	fragments	4	5	6	7
baseline 6k	5,995		21.5			
mb 6k	3,825	2,171	21.4	21.4	21.4	21.4
baseline 12k	11,987		14.0			
mb 12k	7,653	4,335	13.9	13.8	13.8	13.8

- no degradation in performance using less full words

OV Recognition: Fixed-Length Segmentation

- alternative to the morpheme-based segmentation
- main difference: segmentation is done using a g2p model, i.e. considering graphemes and phonemes
- thus: also data-driven and language independent
- fixed length: the maximum value for the context-length L is fixed
- training recipe: basically the same as for the morpheme-based approach

Build G2P Models

`g2p/g2p_model_training_fixedLengthSegmentation.sh`

- for better performance w.r.t. word segmentation, longer graphones have to be allowed:
  `-size-constraint 0,3,0,3`
- now: faster and more memory-efficient configuration necessary
- for all but the first training iteration:
  `-no-emergence`
- best performing model is chosen for the segmentation of OOVs
- (pre-trained g2p models again available online)

[Bisani & Ney 05]

Generate Hybrid LM Text Sources and Vocabulary

`ov-lm-fl/generate_fl_hybrid_lm_text_sources_and_vocabulary.sh`

- uses `makeOvModel.py` provided with the Sequitur G2P toolkit
- replaces the OOVs w.r.t. `recognition.lexicon.${VOCABSIZE}` in the LM text sources by their fragments (graphones)
- fragmentation mapping is also dumped, e.g.:
  `data/ov-lm-fl/train.uniq.text.6k.fragmentmap`
- for vocabulary: discard words which could not be fragmentized
- the top `${VOCABSIZE}` words/fragments are selected for (preliminary) vocabulary: `hybrid-fl.vocab.${VOCABSIZE}`, e.g.
  `data/ov-lm-fl/hybrid-fl.vocab.6k`

Generate Hybrid Recognition Lexicon

`ov-g2p-fl/generate_fl_hybrid_recognition_lexica.sh`

- **vocabulary is separated into full words and fragments**
- **for full words: base dictionary and g2p model are used to generate pronunciations**
- **if both lookup and g2p model fail: discard the word**
- **for fragments: the pronunciations are already there, only formatting issue**
- **for each vocabulary, a R*ASR* lexicon is generated**
- **otherwise: the same as for closed vocabulary/first OV approach**

Generate Hybrid LMs

`ov-lm-fl/ov_lm_fl_training.sh`

- **basically the same than for first OV approach**
- **select best performing LM w.r.t. perplexity on test data**
- **next: start a recognition pass using matching hybrid LM and lexicon**

Joining of Fragments and Scoring

- fragments need to be merged for scoring

```
rms_a0544 1 0.270 0.280 may
rms_a0544 1 0.550 0.180 *an:ae_n*
rms_a0544 1 0.730 0.140 *ti:t_ih*
rms_a0544 1 0.870 0.290 *cip:s_ax_p*
rms_a0544 1 1.160 0.260 *ate:ey_t*
rms_a0544 1 1.420 0.040 @
```

- join consecutive fragments in the generated NIST CTM file

```
rms_a0544 1 0.270 0.280 may
rms_a0544 1 0.550 0.870 anticipate
rms_a0544 1 1.420 0.040 @
```

- `recognition/score.sh`
  - uses `scripts/joinfragments.py`
  - now: use option `-g` to allow for correct merging of fragments
  - joins the fragments in the CTM file before executing sclite

Recognition Results: Open Vocabulary

comparison of systems:

- baseline, closed-vocabulary
- open vocabulary, morpheme-based [mb]
- open vocabulary, fixed-length segmentation [fl]

	vocabulary		WER for n-gram order			
model	words	fragments	4	5	6	7
baseline 6k	5,995		21.5			
mb 6k	3,825	2,171	21.4	21.4	21.4	21.4
fl 6k	2,998	3,002	17.1	17.2	17.2	17.1
baseline 12k	11,987		14.0			
mb 12k	7,653	4,335	13.9	13.8	13.8	13.8
fl 12k	5,995	6,005	10.1	10.2	10.2	10.2

OV Recognition Results: Examples

Vocab. Size	File	Date	Status	Result	
12k	closedvocab_rms_a.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_awb_a.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_clb_a.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_clb_b.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_awb_b.wav	2011-02-03 11:26:49		Open	View
12k	closedvocab_rms_b.wav	2011-02-03 11:26:49		Open	View
12k	baseline_awb_b.wav	2011-02-02 19:16:09		Open	View
12k	baseline_clb_b.wav	2011-02-02 19:16:09		Open	View
12k	baseline_rms_b.wav	2011-02-02 19:16:09		Open	View
12k	baseline_clb_a.wav	2011-02-02 19:16:09		Open	View
12k	baseline_awb_a.wav	2011-02-02 19:16:09		Open	View
12k	baseline_rms_a.wav	2011-02-02 19:16:09		Open	View

- **examples for all three vocabulary sizes (6k, 12k, 24k) for both fragmentation methods presented**
- **fragments have NOT been merged (for better visualization)**

Outline

- Introduction and Goals
- Acoustic Model Training using R*ASR*
- Vocabulary Selection and Pronunciation Generation
- Language Model Training using SRI LM
- Decoding using R*ASR* and Evaluation
- Open Vocabulary Recognition and Evaluation
- Outlook

Further Improvements (1)

- features: incorporate more temporal context by using LDA
- across-word context dependent models
- speaker adaptation
  - segment clustering: use clusters as target classes for adaptation
  - fMLLR: speaker adaptive feature transformation
  - MLLR: transformation of the mean vector in the AM
  - unsupervised adaptation requires multi-pass decoding
- speaker normalization:
  - vocal tract length normalization (VTLN)
    warping of conventional Mel warped filter-bank
- speaker adaptive training (SAT)
  use fMLLR feature transformation in training

Further Improvements (2)

- discriminative training (MPE criterion)
- incorporation of more complex language models
  using lattice rescoring (e.g. for true-casing)
- pronunciation weights
- system combination: incorporate output of multiple ASR systems
- parallelization: use multiple CPUs for AM training and decoding

- all included in R*ASR*
- starting points in the online tutorial
- documentation in the Wiki
- `http://www.hltpr.rwth-aachen.de/rwth-asr/manual`

Demo: Real-Life Systems

quaero

RWTH AACHEN UNIVERSITY

ASR | SMT

Upload | Results | Settings

English | French | German | Polish | Spanish | All

Name:	Sky_News_Top_Story_at_18_11_5th_April_2011.mp4
Language:	English
Duration:	00:02:45
Uploaded:	2011-04-07 13:03:01
Status:	done
Audio:	0:00 2:45
Video:	
Auto Scrolling:	☐

Acknowledgements and Disclaimer

We would like to thank Max Bisani for contributing to the g2p slides.

We thank those who made this Tutorial possible

- Hermann Ney
- Ralf Schlüter
- R*ASR* team

Warning: due to possible changes in the text and audio files offered online, results presented in this tutorial may not be perfectly reproducible!

Thank you for your attention

Stefan Hahn, David Rybach

`rwthasr@i6.informatik.rwth-aachen.de`

`http://www.hltpr.rwth-aachen.de/rwth-asr`

References

[Basha Shaik & El-Desoky Mousa+ 11] M.A. Basha Shaik, A. El-Desoky Mousa, R. Schlüter, H. Ney: Using Morpheme and Syllable Based Sub-words for Polish LVCSR. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Prague, Czech Republic, May 2011. 102

[Beulen & Bransch+ 97] K. Beulen, E. Bransch, H. Ney: State-Tying for Context Dependent Phoneme Models. In *Proc. Fifth Europ. Conf. on Speech Communication and Technology*, Vol. 3, pp. 1179–1182, Rhodes, Greece, Sept. 1997. 36

[Bisani & Ney 05] M. Bisani, H. Ney: Open Vocabulary Speech Recognition with Flat Hybrid Models. In *Interspeech*, pp. 725–728, Sept. 2005. 100, 102, 112

[Bisani & Ney 08] M. Bisani, H. Ney: Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, Vol. 50, No. 5, pp. 434–451, May 2008. 48

[Bridle & Brown 74] J.S. Bridle, M.D. Brown: An Experimental Automatic Word-Recognition System, 1974. 30

[Chen 03] S.F. Chen: Conditional and Joint Models for Grapheme-to-Phoneme Conversion. In *Proc. European Conf. on Speech Communication and Technology*, pp. 2033 – 2036, Geneva, Switzerland, Sept. 2003. 62

[Chen & Goodman 98] S.F. Chen, J. Goodman: An Empirical Study of Smoothing Techniques for Language Modeling. In *Technical Report TR-10-98, Center for Research in Computing Technology*, Harvard University, Cambridge, MA, Aug. 1998. 71, 78

[Deligne & Yvon+ 95] S. Deligne, F. Yvon, F. Bimbot: Variable-Length Sequence Matching for Phonetic Transcription using Joint Multigrams. In *Proc. European Conf. on Speech Communication and Technology*, pp. 2243 – 2246, Madrid, Spain, Sept. 1995. 48, 50

[El-Desoky Mousa & Basha Shaik+ 10] A. El-Desoky Mousa, M.A. Basha Shaik, R. Schlüter, H. Ney: Sub-lexical Language Models for German LVCSR. *IEEE workshop on spoken language technology*, Vol. , pp. 159–164, Dec. 2010. 102

[El-Desoky Mousa & Schlüter+ 10] A. El-Desoky Mousa, R. Schlüter, H. Ney: A Hybrid Morphologically Decomposed Factored Language Models for Arabic LVCSR. In *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies conference*, pp. 701–704, Los Angeles, California, USA, June 2010. 100, 102

[Jiampojamarn & Kondrak 09] S. Jiampojamarn, G. Kondrak: Online Discriminative Training for Grapheme-to-Phoneme Conversion. In *Proceedings of ISCA Interspeech*, pp. 1303–1306, Brighton, U.K., Sept. 2009. 48

[Jiang & Hon$^+$ 97] L. Jiang, H.W. Hon, X. Huang: Improvements on a Trainable Letter-to-Sound Converter. In *Proc. European Conf. on Speech Communication and Technology*, Vol. 2, pp. 605 – 608, Rhodes, Greece, Sept. 1997. 62

[Kanthak & Ney 03] S. Kanthak, H. Ney: Multilingual Acoustic Modeling Using Graphemes. In *European Conference on Speech Communication and Technology*, Vol. 1, pp. 1145–1148, Geneva, Switzerland, Sept. 2003. 23

[Kneser & Ney 95] R. Kneser, H. Ney: Improved Backing-off for M-gram language modeling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 181–184, Detroit, MI, USA, May 1995. 76

[Mermelstein 76] P. Mermelstein: Distance measures for speech recognition, psychological and instrumental. *Pattern Recognition and Artificial Intelligence*, Vol. 116, pp. 374–388, 1976. 30

[Ney & Haeb-Umbach$^+$ 92] H. Ney, R. Haeb-Umbach, B. Tran, M. Oerder: Improvements in Beam Search for 10000-Word Continuous Speech

Recognition. In *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 9–12, San Francisco, CA, USA, March 1992. 87

[Ney & Ortmanns 00] H. Ney, S. Ortmanns: Progress in Dynamic Programming Search for LVCSR. *Proceedings of the IEEE*, Vol. 88, No. 8, pp. 1224–1240, Aug. 2000. 87

[Ortmanns & Ney 00] S. Ortmanns, H. Ney: Look-Ahead Techniques for Fast Beam Search. *Computer Speech and Language*, Vol. 14, No. 1, pp. 15–32, Jan. 2000. 89

[Pagel & Lenzo$^+$ 98] V. Pagel, K. Lenzo, A.W. Black: Letter-to-Sound Rules for Accented Lexicon Compression. In *Proc. Int. Conf. on Spoken Language Processing*, Vol. V, pp. 2015 – 2018, Sydney, Australia, Sept. 1998. 62

[Rybach & Gollan$^+$ 09] D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter, H. Ney: The RWTH Aachen University Open Source Speech Recognition System. In *Proc. Int. Conf. on Speech Communication and Technology*, pp. 2111–2114, Brighton, U.K., Sept. 2009. 12

[Stolcke 02] A. Stolcke: SRILM - An Extensible Language Modeling Toolkit. In *Proc. Int. Conf. on Spoken Language Processing*, Denver, CA, USA, Sept. 2002. 80

[Vozila & Adams+ 03] P. Vozila, J. Adams, Y. Lobacheva, R. Thomas: Grapheme to Phoneme Conversion and Dictionary Verification Using Graphonemes. In *European Conf. on Speech Communication and Technology*, pp. 2469 – 2472, Geneva, Switzerland, Sept. 2003. 62

[Witten & Bell 91] I.H. Witten, T.C. Bell: The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, Vol. 37, pp. 1085–1094, 1991. 74

[Young & Odell+ 94] S. Young, J. Odell, P. Woodland: Tree-Based State Tying for High Accuracy Acoustic Modelling. In *Proc. ARPA Spoken Language Technology Workshop*, pp. 405–410, Plainsboro, NJ, USA, March 1994. 36