Search - Part 2

June 25, 2013
Search - Part 2

• The Search in Automatic Speech Recognition
• DTW review  ➔ pattern based recognition, Optimizations
• Viterbi review  ➔ model based recognition, Optimizations
• Continuous speech recognition
  – Reasons against predicting word boundaries
  – Two level DP
  – One stage DP, Search strategies, stack decoder

• Optimization: How to waste not too much Computation Time
  – Tree-Search, Pruning, Pruning with Beam Search
• Search with LM / Grammar
• Multi-Pass Searches, Problems and Examples
• Producing more than one Hypothesis, Problems
• Speeding up the Search
• Search with Context-Dependent Models
Thoughts

- Two views of time synchronous decoder
  - Expand active states
  - Walk through HMM states and look for best predecessor

- Y-axis can be any shape you want it
  - Can be tree-shaped
  - Can be a grammar (→ finite state transducers)

- But remember decision constraints
  - Decisions can only depend on cumulative score of predecessor state and local transition penalty
Two Strategies for Search Techniques

• All search techniques use two strategies for efficiency:
  – Sharing and Pruning

• **Sharing:**
  Keep intermediate results, so that they can be used by other paths without redundant re-computation

• **Pruning:**
  Disregard unpromising paths without wasting time exploring them further
How to not waste too much Computation Time

• While doing a search (time synchronous or asynchronous), we might often have to compute the same things twice.

Example:

|     | \( p(AE|t) \) | \( p(AEI|t) \) |
|-----|----------------|----------------|
| AEK |                |                |
| K   |                |                |
| KEY |                |                |
| KK  |                |                |
| SH  |                |                |
| AEK |                |                |

• The words "cash" and "can" have the same two phonemes at their beginning. They only differ in their third phoneme.
• So why compute the emission probability for /C/ or /AE/ at the same time frame twice?
• Also: The partial hypothesized alignment path starting at a given frame index is always the same for "cash" and "can". What can we do about it?
Optimization: Tree Search

Let's organize the y-axis as a tree:

- Mark all states that can be the final state of a word
- Expand these final states of a word to roots of successor trees

**Benefit:**
- The maximum number of successor states for any state = number of phonemes, which is usually much smaller (~50) than the number of vocabulary words (>10000).
Optimization: Pruning (Beams)

In general:
Pruning means cutting off a part of the search space that is considered to be unimportant and not to contain the optimal solution that we are looking for.

Remember:
Typical unpruned search spaces have 1,000 time frames and 500,000 HMM states.

Where can we apply pruning?
• Standard approach:
  Do not expand every visited search matrix cell
  – E.g. limit the number of active (i.e. to be expanded) states per frame
  – Or: Decide dynamically which states are expanded (beam search)
Pruning with Beam search

- **Beam search** means:
  Define an "angle" for how much we want to look to the left and right:

- **Typical beam search**: Expand only the states which have accumulated likelihoods that are greater than $b \cdot \text{best likelihood}$. $b=\text{beam}$.

- **How to find a good beam?**
  Trial and error

<table>
<thead>
<tr>
<th>Pruning Method</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Beam search</strong></td>
<td>no sorting, automatic beam width</td>
<td>fuzzy scores $\rightarrow$ wide beam $\rightarrow$ beam $= ?$</td>
</tr>
<tr>
<td><strong>Number of states</strong></td>
<td>constant beam width independent of acoustics</td>
<td>needs sorting</td>
</tr>
</tbody>
</table>
Example: Pruning with one threshold on NAB (Beam ranges from 170 to 230)

Thesis: Monika Woszczyna, pp. 40
Search problems when using Grammars/LMs

- Best predecessor not the same for all words
- Bigrams: Best predecessor depends only on the *identity of current word*:
  - Requires one backpointer per active word-end
  - Still admissible
- Consider how to use bigrams in tree search (delayed bigrams, tree copies)
- Trigrams: best predecessor depends on the *successor of current word*:
  - Things get ugly
  - Lots of state copies, or approximations
- BUT: approximations are better than nothing (poor mans trigrams: just do it)
Language Models / Grammars

- Goal: Estimate $P(W)$ in $P(A|W)*P(W)/P(A)$
- Graphs (very simple tasks)

- Usually:
  - Human-machine applications: Finite state grammars
  - Dictation and human-to-human applications: Statistical Language models (N-grams)
Search with vs. without Language Model (1)

- Grammar defines the possibilities / probabilities of word transitions.
- The transition into an initial state of a word $Z$ is computed by maximizing (Viterbi/DTW) the scores/accumulated distances $C(W)$ of all word-final states in the previous time frame and adding the local acoustic score $am(Z)$.
- This depends on whether we use a grammar or not:

  without grammars  
  
  $X \xrightarrow{c(X)} \: Y \xrightarrow{c(Y)}$ 
  
  with grammars  
  
  $X \xrightarrow{c(X) + am(Z)} \: Y \xrightarrow{c(Y) + am(Z)}$ 

- When using a grammar, we additionally have to add to the accumulated score a language-model score $lm(word, Z)$.
**Search with vs. without Language Model (2)**

- **Without grammar**: The best predecessor state is the same for all word-initial states
  - expand only the word-final state that has the best score in a frame
- **With grammar**: The best predecessor state depends also on the word transition probability/penalty
Tree-Search and Language Model

When using tree search with a language model, we have a problem.

After making a word-to-word transition, we don't know which word we are entering?

So what is the probability of the transition?

**Solution:** delayed bigrams
Delayed Bigrams: Tree-Search with Grammar

We have to "remember" a word-to-word backpointer

➔ When we reach the final state of a word, we still know where we came from.

When entering a new word (root node of the tree) we don't add/multiply the language model immediately, instead we incorporate $p(w_j|w_i)$ when we handle the last state of $w_j$.

Make every word's last state unique. When we process a final state then we know exactly which word we are in and where we came from.

**Disadvantage:** At the entry point we don't know the LM information, this might result in pruning and segmentation errors, especially if LM info is important for task
Tree-structured lexicon:

Problem: No bigram information can be included until word identity is known.

Idea: Estimate unigram information of the remainder subtree (*)

Substitute by the bigram information as soon as possible.

Especially helpful in case of very small beams (less pruning errors)

(Woszczyna/Finke, Steinbiss)

\[
P(\text{have}) + P(\text{has}) \quad \quad \frac{P(\text{has} \mid \text{he})}{P(\text{have}) + P(\text{has})} \quad \frac{P(\text{have} \mid \text{I})}{P(\text{have}) + P(\text{has})}
\]

(*) with really clever setup instead distribute N-gram estimate
Multi-Pass Searches

Remember forward-backward algorithm:
First pass: forward pass to compute α,
Second pass: backward to compute β

Why use multiple passes in continuous speech recognition?
• Stack decoder: We could use some good estimator for the score of the remainder (A*)
• Pruning: Good lookahead
  ➔ Decide which part of the search space should be pruned away.
• Recover from errors resulting from delayed bigrams

Forward-backward search:
• First run backward pass
• Then run forward pass using backward scores to do pruning
• Not applicable for run-on!
Problems with Multi-Pass Searches

- When using a search pass to compute information for pruning, this pass must be much faster than the actual search pass.
- By definition, it can not produce better results than the actual search pass (otherwise make it the actual pass).
- What if we want/need \textit{runon} recognition (i.e. start recognizing - and processing - before the speaker has finished the sentence).
- We can not wait till the end of the utterance to compute an estimate for the remainder.
- How do we run a backward pass for continuous speech recognition with grammar?
- If we know $p(w_j \text{ follows } w_i)$ we don't automatically have $p(w_i \text{ precedes } w_j)$.
- Where do we get that \textit{backward bigrams} from?
- Have to know unigrams and apply bayes rule:

\[
P(B|A) = P(A|B) \cdot P(B) / P(A)
\]
Example for a Multi-Pass Search

First Pass:
Run a tree search (forward direction).
Use a narrow beam and/or weak but fast acoustic models.
Remember for every word, at what times it was "active" (not pruned away).

Result: smaller search space =

<table>
<thead>
<tr>
<th>word 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>word 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>word n</td>
<td></td>
</tr>
</tbody>
</table>

Second Pass:
Run a regular Viterbi search (one-stage DTW) but consider the inactive areas already pruned away. This time use a wider beam and/or powerful but possibly slower acoustic models.
Producng more than one Hypothesis (1)

Reasons why the recognizer should also deliver less likely hypotheses:

- Do postprocessing on all hypotheses with additional knowledge ...
- E.g. According to pragmatic knowledge "gimme a nudist play" is less likely than "gimme a new display"
- Parsing on the best parts of several different hypotheses such that the whole utterance can be parsed (speech understanding)
- Offer error recovery mechanisms to the user, e.g.:
Producing more than one Hypothesis (2)

How can the recognizer produce less likely hypotheses:

**Isolated word recognition:**
- Easy: Do not only report the word with the highest likelihood but also the next \( n \) best words

**Continuous speech recognition:**

- **Different recognizers:**
  - Run several different recognizers, report all results
  - Nice side effect: DARPA's ROVER takes majority vote
  \( \Rightarrow \) 10 - 20% error reduction

- **Single recognizer:**
  - Let Viterbi not only remember best predecessor but best \( k \) predecessors
  - Produce multiple backtraces (theoretically up to \( k^T \)), so we need pruning again for finding "good" backtraces
Problems with \textit{n}-best Hypotheses

- Often non-content words (a, the, in, of) are difficult to recognize
- Typical \textit{n-best} output: show me in her face please
  show the inner face see
  show the in her face please
  show the in her face see
  show me the interface please

$\rightarrow$ Many irrelevant variations but wrong content word does not change
  - \textit{n} too small $\rightarrow$ Little use (correct hypo not among \textit{n} best)
  - \textit{n} too large $\rightarrow$ System can become slow and clumsy,
    slow search for correct hypothesis

- Solution: \textbf{Word lattices}
Output Formats

Different types of output:

- 1\(^{st}\)-best: Word string
- N-best: N word strings
- Lattices: Time-marked directed, acyclic graph
- Confusion networks: Directed graph with total ordering
Output Formats: Word Lattices

- Lattices nodes contain:
  - Word identities
  - Acoustic scores
  - Time information

- Lattice links contain:
  - Acoustic score for context models
  - LM scores (computed on-the-fly)

- Generate $1^{st}$-best, N-best, and CNs from lattices
Output Formats: Example

- Lattice:

  - Contains:

    Node = word, begin + end frame, ac-score ($\alpha, \beta, \gamma$)

    → = ac-delta-score ($lm$-score)
Output Formats: Confusion Networks

• Sketch of construction algorithm
  – Prune lattice using (lattice link) confidences
  – Collect ordering information
    • Words have temporal order
    • Words are linked
    • Ordering prevents clustering
  – Construct within-word clusters using confidence
  – Construct inter-word clusters using confidence
  – Order output according to item confidence
Example: Lattice vs Confusion Network (CN)

Output Formats: Example (1)

- Original decoder output: Word-Lattice

```
so <uhm> that clear
dad is
```

- Back-tracking:
  - 1<sup>st</sup>-best: so is dad clear
  - 2<sup>nd</sup>-best: so <uhm> that clear
Output Formats: Example (2)

- Original decoder output: Word-Lattice

  so \[\rightarrow\] is \[\rightarrow\] dad \[\rightarrow\] clear
  \[\leftarrow\] <uhm> \[\rightarrow\] that

- Confusion Network without time marks:

  so \[\rightarrow\] is \[\rightarrow\] dad \[\rightarrow\] clear
  \[\leftarrow\] <uhm> \[\rightarrow\] that

- New hypo: so is that clear
Output Formats

- **1st-best** is “baseline” for WER (word error rate)

- **N-best** can have lower error rate, but often difficult to handle

- **Lattices** can have up to 1/3rd “lattice error rate”, but how to find the path with lowest WER

- **Confusion networks** usually have lower WER (~1%) and are a more compact representation → “best output”?
Confidence Measures

• Confidences can be associated with every item of a hypothesis, lattice, CN, ...
  – Useful for judging the output of a recognizer
  – Example of *.ctm hypo file with confidences:

  • # LDC_20011121-1700_d*_NONE-A-0188 1777.970 0.779050
  • LDC_20011121-1700_d*_NONE A 1780.97 0.11 IT 0.65
  • LDC_20011121-1700_d*_NONE A 1781.09 0.14 WOULD 0.92
  • LDC_20011121-1700_d*_NONE A 1781.23 0.14 COME 1.00
Confidence Measures - Generation

• Confidence measures can be derived from many features:
  – Lattice Link probability (the “standard” way)
  – Hypothesis density
  – Acoustic stability
  – …

Confidence Measures: Example

- Example of lattice link “gamma” confidence measure
Combination of Recognizers

• Want to merge output from different recognizers
  – Reduce WER
  – Improve confidences

• Approaches:
  – ROVER (Jon Fiscus)
  – Lattice intersection
  – Confusion Network Combination

• Works for different channels or recognizers
Speeding up the Search

Therea are several ways to make the search faster:

Reduce number of searched states:
• Pruning techniques (beam search)
• Multiple passes (reduce search to active words)
• Lookaheads: fast predictor for likelihood of current
  (phoneme/word) → prune entire word if unlikely
• Language model lookahead: Don't expand states into
  words that are not likely according to language model

Reduce computation effort per state:
• High degree of tying
  → compact tree, short state-axis, fewer emission probabilities
• Presearched / hierarchically organized codebook vectors
  → Faster calculation of emission probabilities
Isolated word recognition
Easy: only the word-HMMs look a bit more complicated

Continuous speech recognition
Example: Simple search space for a two words CSR.

Lexicon:

<table>
<thead>
<tr>
<th>word</th>
<th>pronunciation</th>
</tr>
</thead>
<tbody>
<tr>
<td>what</td>
<td>W O T</td>
</tr>
<tr>
<td>what</td>
<td>H W O T</td>
</tr>
<tr>
<td>had</td>
<td>H A D</td>
</tr>
<tr>
<td>had</td>
<td>H A T</td>
</tr>
</tbody>
</table>
The search tree becomes less compact

⇒ larger search space when we use context dependent models

Only words that start with the same polyphone share a common root. Cross-word context dependence is very complicated.

⇒ restrict it to e.g.:

• only first and last phonemes of a word are modeled dependent on neighboring words
• the maximum context width can go only one phoneme into the neighboring word
• one-phoneme words are treated separately
Search with context dependent phonemes

- What models to use for last phoneme:
  - Don’t know successor word when evaluating last phone in word
  - Keep multiple copies of last phonemes (fortunately not many word ends active)

- What models to use for first phoneme:
  - With delayed n-grams, predecessor word unknown while evaluating first phone
  - Keep copies of first phone for all varieties, transit from best into second phone, then when resolving delayed n-gram fix score by incorporating score difference.

- This makes first phonemes (almost always active) and last phonemes the most expensive parts of time synchronous decoding
Phoneme Lookaheads

- Idea: Acoustic Lookaheads at the phoneme level
- Using simple (fast) acoustic models for phonemes
- Predict how well each partial hypotheses will do in the next couple of frames
- Use the estimated future score to prune bad hypotheses before more costly score calculations are computed
Search Summary (Part 1+2)

• The Search in Automatic Speech Recognition
• DTW review \(\Rightarrow\) pattern based recognition, Optimizations
• Viterbi review \(\Rightarrow\) model based recognition, Optimizations
• Continuous speech recognition
  – Reasons against predicting word boundaries
  – Two level DP
  – One stage DP, Search strategies, stack decoder

• Optimization: How to waste not too much Computation Time
  – Tree-Search, Pruning, Pruning with Beamsearch
• Search with LM / Grammar
• Multi-Pass Searches, Problems and Examples
• Producing more than one Hypothesis, Problems
• Speeding up the Search
• Search with Context-Dependent Models
Thanks for your interest!