No.9

Hypothesis Search in Large Vocabulary ASR

Prof. Hui Jiang
Department of Computer Science and Engineering
York University

Automatic Speech Recognition (III): Search for LVCSR

Prof. Hui Jiang
Department of Computer Science and Engineering
York University, Toronto, Canada
hj@cse.yorku.ca
ASR Solution

\[
\hat{W} = \arg \max_{W \in \Gamma} p(W \mid X) = \arg \max_{W \in \Gamma} P(W) \cdot p(X \mid W)
\]

\[
= \arg \max_{W \in \Gamma} \overline{P}_T(W) \cdot \overline{p}_A(X \mid W)
\]

- \(\overline{P}_A(X \mid W)\) — **Acoustic Model (AM)**: gives the probability of generating feature \(X\) when \(W\) is uttered.

- \(\overline{P}_T(W)\) — **Language Model (LM)**: gives the probability of \(W\) (word, phrase, sentence) is chosen to say.

---

Overview of Statistical ASR

[Diagram of ASR process]

- Front End Parameterisation
- Acoustic Models
- Parameterised Speech Waveform
- Pronouncing Dictionary
- Language Model: \(P(W) \cdot P(Y \mid W)\)
How to postulate word sequence?

- First thought: enumerate all possible word sequences one by one
  - Expand into a large composite HMM
  - Calculate the score and look for the best sequence
  - Impossible even for small vocabulary task, e.g., digit string.
- Solution: build an overall recognition network accommodating all possible word sequence → search for the best path
  - Consider the task grammar and the language modeling constraints (FSG, n-gram, context-free)
  - Build search network based on the task grammar
  - Expand into a single huge composite HMM
  - Given a speech feature sequence, use the Viterbi algorithm to search for the best alignment path through the network.
  - The alignment path → the most likely word sequence (output)
  - Each alignment path corresponds to one word sequence; but each word sequence has many possible alignment paths.
    - Viterbi Approximation → easy implementation

Search Space Representation

- Postulating word sequences is a typical search problem in CS.
- First of all, how to specify search space in ASR?
- Obviously, the search space depends on the underlying grammar.
- In ASR, language grammar is given in the following forms:
  - Finite State Grammar (FSG):
    Applications: voice dialing, digit string recognition, etc.
  - N-gram: uni-gram, bi-gram, tri-gram, 4-gram
    Applications: Dictation system, broadcast news transcription, etc.
  - Context-free Grammar (CFG) → recursive transition network
    CFG is convenient to refer to high-level task-specific concepts, such as dates, names, inquiry patterns, etc.
    Useful in speech understanding
Search Space(1): FSG

- FSG itself is a search network; directly expand into composite HMM based on lexicon and acoustic models.

(a) single digit
(b) single digit with start/end silence
(c) Digit string with start/end silence
(d) Digit string with optional silence

Search Space(2): Unigram

- Word-loop network is sufficient for unigram LM.
Search Space(3): Bigram

Network for bi-gram is a bit complex; need more glue nodes.

Search Space(4): Back-off Bi-gram

If back-off bi-gram is used, glue nodes can be merged for back-off contexts to reduce links. (used in HTK)
Search Space(5): Back-off Bigram LM with WFST

- No full context in back-off n-gram LM.
  - Observed context: use n-gram condition probabilities.
  - Unobserved context: back-off to lower level n-1 gram.
- WFST for back-off bi-gram LM:

![WFST diagram for back-off bi-gram LM]

Search Space(6): Trigram

- Network for tri-gram becomes significantly complicated.
- Network example for 2-word \( (w_1, w_2) \) vocabulary

![WFST diagram for trigram LM]
Search Space(7): Back-off Trigram

- Representation of a full trigram LM for large vocabulary is prohibitive.

- It is possible to represent a back-off trigram LM even for very large vocabulary.

- WFST example ...

Search Space(8): Back-off Trigram

- One back-off node for bigram
- Multiple back-off nodes for trigrams (one for each history)
Token Passing (1): simple implementation model for Viterbi decoding

- For a large or even medium size HMM, hard to maintain 2-D trellis to implement the Viterbi decoding algorithm.
- Token passing paradigm: equivalent; easy to implement for large HMM's.
- Token passing:
  - Each HMM state holds a movable token which contains all info about its partial travel from a HMM start state up to the current state, e.g. partial log prob $\delta(.)$ and the partial path.
  - Viterbi search becomes a token propagation process.

Token Passing Algorithm

- Initialization: each HMM initial state holds a token with value 0;
- Propagation:
  - For each observation feature vector $o_t$, $t=1,2,..., T$
    - For each HMM state $i$ do
      1. Pass a copy of the token in state $i$ to all connecting states $j$ by following HMM state transition; updating value of the new tokens by $a_{ij} + b_{ij}(o_t)$;
      2. Discard the original tokens;
    End
  - For each HMM state $i$ do
    - if more than one tokens enter state $i$, keep the best one, discard the rest;
  End
  End
- Termination:
  - Examine all final states, the token with the best value passed the best path; its value $\rightarrow$ Viterbi score; recover path.
Token Passing Example

Token Passing: record boundaries

Recording Word Boundary Decisions
Techniques to Accelerate Search in ASR

- Beam search
  - Prune unlikely candidates at the earliest stage.
- Fast-match
- Tree-organized pronunciation lexicon
  - For data sharing and better pruning strategy.
  - How to construct search space for tree lexicon?
  - Language Model Look-Ahead: how to apply LM earlier?
- One-pass search vs. Multi-pass search
  - Integrated one-pass search: integrate all available knowledge sources and explore the whole search space once; slow.
  - Multi-pass search: use partial knowledge (e.g., simpler models) to reduce search space; explore the reduced search space by more complicated models; fast.
- Dynamical network expansion
- Static decoding based on minimized WFST
- Alternative outputs:
  - N-Best list: how to generate?
  - Word-graph: compact representation of more candidates.

Beam Search (I)

- Beam Search: every time frame, the best score in all partial paths (tokens in token passing) is noted and any partial paths (tokens) whose score lies more than a beam-width below this best score is pruned from further consideration.
- Instead of searching for the entire dark room, just follow the beam of your flashlight.
- Beam-width is a pre-set constant to control the degree of pruning.
- Beam search makes the prohibitive search problem feasible.
- In beam search, search space never goes out of control.
Beam Search (II)

- Beam search is THE most important pruning strategy to accelerate search in speech recognition.
- Beam search is not admissible: it may miss the best path; but this seldom happens in practice if the beam-width is set properly.

Beam Search (III)

- Acoustic pruning: retain only hypotheses with a score close to the best hypothesis for further consideration.
  - Regular beam search for all in-HMM partial candidates.
  - Acoustic beam-width $P_a$.
- Language model pruning (word ending pruning):
  - The optimal path seems more stable at the word-ending points during the search especially after applying LM scores.
  - More aggressive pruning is possible at word-end.
  - Word-ending (LM) beam-width $P_{LM}$. ($P_{LM}$ can be chosen to smaller than $P_a$ to ward off more unlikely candidates)
- Histogram Pruning:
  - Each time, instead of setting a beam width, survive only the best $N$ candidates.
  - Sorting is prohibitive; usually implement by histogram.
ASR Search Algorithms

· Dynamic search network expansion
  – Tree lexicon
  – Language model look-ahead
  – Dynamic expansion

· Static optimized network
  – Static back-off LM network
  – Expansion using WFST composition
  – Optimization using WFST determinization and minimization

Tree Lexicon Organization

· Linear lexicon: each word in vocabulary is modeled separately:
  – Essentially, it is a linear sequence of phonemes according to its pronunciation.

· Tree lexicon: all words in vocabulary can be organized into a prefix tree:
  – Better data sharing; more effective pruning.
  – Each leaf node represents one word.
  – Extremely important for large vocabulary cases.
Tree Lexicon: problems

- Problems with a tree lexicon:
  - The identity of the hypothesized word is unknown until reaching a leaf node.
    - Language model (LM) scores can’t be applied until at the end of tree → ineffective pruning in beam search
  - Search space is hard to formulate unless making lots of tree copies.
- Conceptual example:
  - Three words in vocabulary
  - A network for only 2-word sentences
  - For bi-gram: introducing merging nodes for previous word
  - For tri-gram: introducing merging nodes for previous two words

Search Space for Tree Lexicon:
Language Model Look-ahead

- In tree lexicon, can’t apply LM score due to unknown id of current word.
- Better to incorporate LM knowledge as soon as possible to prune those unlikely candidates in grammar.
- LM look-ahead: apply maximum LM scores of all words which can be reached from the current node.

Dynamic Network Expansion
How to handle huge search space in large vocabulary

- Fast Match: phoneme look-ahead
  - Look-ahead some feature vectors to determine a small set of most likely phoneme from the current time point.

- Multiple-pass search strategy:
  - 1st pass: use simple language model (unigram, bi-gram) to reduce search space.
  - 2nd pass: use more complicated model (such as tri-gram) to search for the result only in the above reduced space.

- Single-pass search strategy:
  - Dynamic network expansion:
    - No a whole static network is built beforehand (too big).
    - Expand the net dynamically during the search process.

Static Optimization Network using WFST

- Build a huge static search network from LM: Composition
  - LM-based Grammar WFST (G)
  - Pronunciation Lexicon (L)
  - Context-Dependency Transducer (C)
  - Sub-word HMM (H)

\[ F = H \circ C \circ L \circ G \]

- Compact the network using graph algorithms.
  - Determinization
  - Minimization

\[ \text{min( det(F) )} \]
Context Dependency

WFST for ASR
WFST for ASR

\[ \det(\tilde{L} \circ G) \]

\[ \min(\det(\tilde{L} \circ G)) \]
WFST for ASR

\[ \text{push}(\pi_z(\text{det}(L \circ G))) \]

WFST for Speech Recognition

<table>
<thead>
<tr>
<th>network</th>
<th>states</th>
<th>transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G )</td>
<td>1,339,664</td>
<td>3,928,010</td>
</tr>
<tr>
<td>( L \circ G )</td>
<td>8,606,729</td>
<td>11,408,721</td>
</tr>
<tr>
<td>( \text{det}(L \circ G) )</td>
<td>7,082,404</td>
<td>9,836,629</td>
</tr>
<tr>
<td>( C \circ \text{det}(L \circ G) )</td>
<td>7,273,035</td>
<td>10,201,269</td>
</tr>
<tr>
<td>( \text{det}(H \circ C \circ L \circ G) )</td>
<td>18,317,359</td>
<td>21,237,992</td>
</tr>
<tr>
<td>( F )</td>
<td>3,188,274</td>
<td>6,108,907</td>
</tr>
<tr>
<td>( \text{min}(F) )</td>
<td>2,616,948</td>
<td>5,497,952</td>
</tr>
</tbody>
</table>

Table 1: Size of the first-pass recognition networks in the NAB 40,000-word vocabulary task.

<table>
<thead>
<tr>
<th>network</th>
<th>x real-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C \circ L \circ G )</td>
<td>12.5</td>
</tr>
<tr>
<td>( C \circ \text{det}(L \circ G) )</td>
<td>1.2</td>
</tr>
<tr>
<td>( \text{det}(H \circ C \circ L \circ G) )</td>
<td>1.0</td>
</tr>
<tr>
<td>( \text{min}(F) )</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table 2: Recognition speed of the first-pass networks in the NAB 40,000-word vocabulary task at 83% word accuracy
Weighted Finite State Transducer (WFST)

- WFST: weighted finite state transducer (or acceptor):

```
WFST Operations

- Composition: $C = A \circ B$

- Determinization: $D = \text{det}(C)$
  - deterministic automaton: every state has at most one out-going transition with any given label.

- Re-weighting (Weight pushing): $E = \text{push}(D)$

- Minimization: $F = \text{min}(E)$
```
WFST Operations: Examples

Multiple Outputs

- How to generate a short list of multiple outputs instead of a single best?
  - To apply more knowledge to pick up one.

- N-Best List:
  - A list of top N best candidates

- Word graph:
  - A compact representation of a large number of candidates.

- How to generate N-best list or word graph from search process?
  - Standard Viterbi search can find the best one.
  - Modify the Viterbi somewhat for this feature.
N-Best List: example

<table>
<thead>
<tr>
<th>Rank</th>
<th>Hypotheses</th>
<th>Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SILENCE HARD ROCK SILENCE</td>
<td>-5690.11</td>
</tr>
<tr>
<td>2</td>
<td>SILENCE HARD WRONG SILENCE</td>
<td>-5603.17</td>
</tr>
<tr>
<td>3</td>
<td>SILENCE HARD RAW SILENCE</td>
<td>-5606.32</td>
</tr>
<tr>
<td>4</td>
<td>SILENCE A HARD ROCK SILENCE</td>
<td>-5620.58</td>
</tr>
<tr>
<td>5</td>
<td>SILENCE HARD ROT SILENCE</td>
<td>-5622.06</td>
</tr>
<tr>
<td>6</td>
<td>SILENCE HARD RON SILENCE</td>
<td>-5623.69</td>
</tr>
<tr>
<td>7</td>
<td>SILENCE CARD WRONG SILENCE</td>
<td>-6624.51</td>
</tr>
<tr>
<td>8</td>
<td>SILENCE CARD RAW SILENCE</td>
<td>-6625.66</td>
</tr>
<tr>
<td>9</td>
<td>SILENCE YOU HARD ROCK SILENCE</td>
<td>-6628.96</td>
</tr>
<tr>
<td>10</td>
<td>SILENCE HART WRONG SILENCE</td>
<td>-6629.92</td>
</tr>
<tr>
<td>11</td>
<td>SILENCE HEART WRONG SILENCE</td>
<td>-6630.82</td>
</tr>
<tr>
<td>12</td>
<td>SILENCE ARE HARD ROCK SILENCE</td>
<td>-6635.11</td>
</tr>
<tr>
<td>13</td>
<td>SILENCE CARD ROCK SILENCE</td>
<td>-6636.06</td>
</tr>
<tr>
<td>14</td>
<td>SILENCE OF HARD ROCK SILENCE</td>
<td>-6637.56</td>
</tr>
<tr>
<td>15</td>
<td>SILENCE CARD ROT SILENCE</td>
<td>-6641.32</td>
</tr>
<tr>
<td>16</td>
<td>SILENCE CARD RON SILENCE</td>
<td>-6643.03</td>
</tr>
<tr>
<td>17</td>
<td>SILENCE A HARD WRONG SILENCE</td>
<td>-6645.74</td>
</tr>
<tr>
<td>18</td>
<td>SILENCE PART WRONG SILENCE</td>
<td>-6646.36</td>
</tr>
<tr>
<td>19</td>
<td>SILENCE HART ROT SILENCE</td>
<td>-6646.85</td>
</tr>
<tr>
<td>20</td>
<td>SILENCE A HARD RAW SILENCE</td>
<td>-6648.89</td>
</tr>
</tbody>
</table>

True Transcription: hard rock

Word Graph (Lattice): example (1)
Word Graph: example (2)

Other search strategies:

- Viterbi algorithm: time-synchronous breadth-first search

- Depth-first: A* search (or stack decoding)
  - Time-asynchronous search
  - Expend and evaluate partial hypothesis from a stack.
  - Widely used in AI search.
  - Admissible: the best path is guaranteed as long as the heuristics are not over-estimated.
  - Not popular anymore in speech recognition.
  - NO TIME to cover.