+ FSAs: A recurring structure in speech

Phonetic HMM

Viterbi trellis

Language model

Pronunciation modeling
The language of sheep: /baa+!/

- We can say the following things about this machine:
  - It has 5 states
  - b, a, and ! are in its alphabet
  - q₀ is the start state
  - q₄ is an accept state
  - It has 5 transitions
FSM as an Input tape acceptor

Given an input “tape”, does my machine accept or reject that input?

Transition Table

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>a</th>
<th>!</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Non-Determinism**

Stay in q2 or go to q3?

Input: b a a

Both define /baa+/
Non-Determinism cont.

- Yet another technique
  - Epsilon transitions
  - Key point: these transitions do not examine or advance the tape during recognition
Equivalence

- Non-deterministic machines can be converted to deterministic ones with a fairly simple construction
  - That means that they have the same power:
  - non-deterministic machines are not more powerful than deterministic ones in terms of the languages they can accept

- Two basic approaches to ND recognition (used in all major implementations of regular expressions)
  - Either take a ND machine and convert it to a D machine and then do recognition with that.
  - Or explicitly manage the process of recognition as a state-space search (leaving the machine as is).
Non-Deterministic Recognition: Search

- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.

- But not all paths directed through the machine for an accept string lead to an accept state.

- No paths through the machine lead to an accept state for a string not in the language.

- Non-determinism doesn’t get us more formal power and it causes headaches so why bother?
  - More natural (understandable) solutions
Example

Brandeis CS114 2013 Meteer
Compositional Machines

- Formal languages are just sets of strings
- Therefore, we can talk about various set operations (intersection, union, concatenation)
- This turns out to be a useful exercise
Weighted finite state acceptors

- Like a normal FSA but with costs on the arcs and final-states
  - Note: cost comes after “/”. For final-state, “2/1” means final-cost 1 on state 2.

- View WFSA as a function from a string to a cost.

- In this view, unweighted FSA is \( f : \text{string} \rightarrow \{0, \infty\} \).

- If multiple paths have the same string, take the one with the lowest cost.

- This example maps \( ab \) to \((3 = 1+1+1)\), all else to \( \infty \).

Thanks for Mirko Hannemann for this slide
Weights vs. costs

- Use “cost” to refer to the numeric value, and “weight” when speaking abstractly, e.g.:
  - The acceptor above accepts a with unit weight.
  - It accepts a with zero cost.
  - It accepts bc with cost $4 = 2 + 1 + 1$
  - State 1 is final with unit weight.
  - The acceptor assigns zero weight to xyz.
  - It assigns infinite cost to xyz.

Thanks for Mirko Hannemann for this slide
Combinations

- **Union:** parallel
- **Concatenation:** series
- **Kleene closure:** arbitrary repetition
WSFAs in speech

Language modeling

Pronunciation modeling

Figure 1: Weighted finite-state acceptor examples. By convention, the states are represented by circles and marked with their unique number. The initial state is represented by a bold circle, final states by double circles. The label $l$ and weight $w$ of a transition are marked on the corresponding directed arc by $l/w$. When explicitly shown, the final weight $w$ of a final state $f$ is marked by $f/w$.
WFS Transducer

- Accept an input while producing an output

Input phonemes: output words

Diagram of WFS Transducer with input, output, and weight labels:

Input:
- d: data/1
- d: dew/1

Output:
- a:b/0.3

Weights:
- a:b/0.1
- b:a/0.2
- b:b/0.5
- a:a/0.4
- c:a/0.3
- 3/0.6

States:
- 0
- 1
- 2
- 3
- 4
- 5
- 6
**FST for morphology:** Foxes and Cats

\[ \begin{align*}
\text{Lexical} & \quad f \quad o \quad x \quad +N \quad +Pl \\
\text{T}_{\text{lex}} & \quad 0 \quad 1 \quad 2 \quad 5 \quad 6 \quad 7 \\
\text{Intermediate} & \quad f \quad o \quad x \quad ^\wedge \quad s \quad # \\
\text{T}_{\text{e-insert}} & \quad 0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 0 \\
\text{Surface} & \quad f \quad o \quad x \quad e \quad s
\end{align*} \]
Composition

The weighted composition algorithm generalizes the classical state-pair construction for finite automata to the weighted case.

Given two transducers \( T_1 \) and \( T_2 \), the composition \( T_1 \circ T_2 \) is computed from the weights of the two corresponding paths in \( T_1 \) and \( T_2 \). In particular, the composition of context-dependent transducers is conveniently and efficiently computed by mapping the local states of \( T_1 \) to the final states of \( T_2 \).

The composition operation is the sum. More generally, the weights represent log probabilities or negative log probabilities, as is common in ASR for numerical stability. The operation is the sum.

Example:

\[
\begin{align*}
\text{Word transducer 1:} & \quad 0 \rightarrow 1: a:b/0.1, c:a/0.3 \\
& \quad 1 \rightarrow 2: a:a/0.4, b:b/0.5 \\
& \quad 2 \rightarrow 3/0.6 \\
\text{Word transducer 2:} & \quad 0 \rightarrow 1: b:c/0.3 \\
& \quad 1 \rightarrow 2: a:b/0.4 \\
& \quad 2 \rightarrow 3/0.7 \\
\end{align*}
\]

Composition:

\[
\begin{align*}
& \text{Path 1:} \quad 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \\
& \quad \text{Weights:} \quad 0.1 + 0.3 + 0.5 = 0.9 \\
& \text{Path 2:} \quad 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \\
& \quad \text{Weights:} \quad 0.3 + 0.4 + 0.7 = 1.4 \\
\end{align*}
\]

Resulting weight:

\[
0.9 + 1.4 = 2.3
\]

The composition is done with the same operation for each pair of paths, the resulting weight is the sum of the weights of the two corresponding paths.
WFSTs in Action with OpenFST

- Create text file
- Compile
- Print
- Show info
- Union
- Concatenate
- Compose
- Invert