Learning Weighted Finite-State Transducers

SPFLODD

October 27, 2011
Background

• This lecture is based on a paper by Jason Eisner at ACL 2002, “Parameter Estimation for Probabilistic Finite-State Transducers.”
  – This is perhaps the most under-appreciated paper in the past ten years of NLP.

• Full disclosure: Jason Eisner was my Ph.D. advisor.
  – He’s one of the smartest people I have ever met.
# Finite-State Automaton

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>finite set of states</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>finite vocabulary</td>
</tr>
<tr>
<td>$q_0 \in Q$</td>
<td>start state</td>
</tr>
<tr>
<td>$F \subseteq Q$</td>
<td>set of final states</td>
</tr>
<tr>
<td>$\delta : Q \times \Sigma^* \rightarrow 2^Q$</td>
<td>transition function; possible next states given current state and input symbol(s)</td>
</tr>
</tbody>
</table>
## Finite-State Automaton (Maybe Better?)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>finite set of states</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>finite vocabulary</td>
</tr>
<tr>
<td>$q_0 \in Q$</td>
<td>start state</td>
</tr>
<tr>
<td>$F \subseteq Q$</td>
<td>set of final states</td>
</tr>
<tr>
<td>$A \subseteq Q \times \Sigma^* \times Q$</td>
<td>set of transitions (source state, symbol sequence, target state)</td>
</tr>
</tbody>
</table>
Finite-State Automaton

• Automaton that recognizes a regular language
  – Key transformations: remove $\epsilon$-transitions, determinize, minimize
• Implementation of a regular expression
• Regular languages are closed under numerous operations
  – Concatenation, union, intersection, Kleene $\ast$, difference, reverse, complement, ...
• Correspond to regular grammars (type 3 in Chomsky hierarchy)
• Pumping lemma: necessary condition for a language to be regular
FSA as a Dictionary

• Example: 850 words in “Basic English”
• Each word is an FSA
Ten-Word Dictionary
Remove $\varepsilon$-transitions
Determinize
Minimize
# Full 850-Word Dictionary

<table>
<thead>
<tr>
<th>Operation</th>
<th>states</th>
<th>final states</th>
<th>arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>5303</td>
<td>850</td>
<td>5302</td>
</tr>
<tr>
<td>Remove $\varepsilon$-transitions</td>
<td>4454</td>
<td>850</td>
<td>4453</td>
</tr>
<tr>
<td>Determinize</td>
<td>2609</td>
<td>848</td>
<td>2608</td>
</tr>
<tr>
<td>Minimize</td>
<td>744</td>
<td>42</td>
<td>1535</td>
</tr>
</tbody>
</table>
Generalizations

• Finite-state recognizer is a function from $\Sigma^* \rightarrow \{0,1\}$
  – Meaning: $\text{fsa}(s) = 1 \iff s$ is in the language

• Other rational relations ...
  – Finite-state transducer: $\Sigma^* \rightarrow \Delta^*$
  – Weighted FSA: $\Sigma^* \rightarrow \mathbb{R}$
  – Weighted FST: $\Sigma^* \rightarrow \Delta^* \times \mathbb{R}$

• WFSAs and WFSTs can be considered probabilistic (but don’t have to be)
Relations on Strings

• A relation is a set of \( (input, output) \) pairs.
  – More general than \textit{functions} because you can represent ambiguity and optionality!
  – For standard FSAs, think of input = output.

• Rational relations are a special kind of relation with a wide range of \textit{closure} properties.

• Rational relations can be understood as a declarative programming paradigm:
  – source code is a regular expression
  – object code is a 2-tape automaton called a \textbf{finite-state transducer (FST)}
  – optimization is accomplished by determinization and minimization
  – supports nondeterminism, parallel processing over infinite sets of input strings, reverse computation from outputs to inputs
# Finite-State Automata and Transducers

<table>
<thead>
<tr>
<th>FSA</th>
<th>FST</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td></td>
<td>finite set of states</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td></td>
<td>finite (input) vocabulary</td>
</tr>
<tr>
<td>$\Delta$</td>
<td></td>
<td>finite output vocabulary</td>
</tr>
<tr>
<td>$q_0 \in Q$</td>
<td></td>
<td>start state</td>
</tr>
<tr>
<td>$F \subseteq Q$</td>
<td></td>
<td>set of final states</td>
</tr>
<tr>
<td>$\delta : Q \times \Sigma^* \rightarrow 2^Q$</td>
<td></td>
<td>transition function; possible next states given current state and input symbol(s)</td>
</tr>
<tr>
<td>$\delta : Q \times \Sigma^* \times \Delta^* \rightarrow 2^Q$</td>
<td></td>
<td>... and output symbol(s)</td>
</tr>
</tbody>
</table>
Eisner’s Running Example
Weighted Relations

• Assign **scores** to (input, output) pairs.
  – Sometimes interpreted as $p(\text{input}, \text{output})$
  – Sometimes interpreted as $p(\text{output} \mid \text{input})$
  – Sometimes neither

• This idea unifies many NLP approaches:
  – sequence labeling
  – chunking
  – normalization
  – segmentation
  – alignment
  – speech recognition (Pereira and Riley, 1997)
  – machine translation (Knight and Al-Onaizan, 1998)
# Weighted FSTs

<table>
<thead>
<tr>
<th>FST</th>
<th>Weights</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td></td>
<td>finite set of states</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td></td>
<td>finite input vocabulary</td>
</tr>
<tr>
<td>$\Delta$</td>
<td></td>
<td>finite output vocabulary</td>
</tr>
<tr>
<td>$q_0 \in Q$</td>
<td></td>
<td>start state</td>
</tr>
<tr>
<td>$F \subseteq Q$</td>
<td>stop weights</td>
<td>set of final states</td>
</tr>
<tr>
<td>$\delta : Q \times \Sigma^* \times \Delta^* \rightarrow 2^Q$</td>
<td>arc weights</td>
<td>transition function; possible next states given current state and input symbol(s) and output symbol(s)</td>
</tr>
</tbody>
</table>
WFSTs

FSTs

FSAs

WFSAs

weights are in \{0, 1\}

represent sets (languages), a special kind of relation where output = input
Weights and Scores

• Weights are assigned to transitions and to ending a path in each state.

• **Score** of a path is the product of the transition weights and the stop weight.

• “Zero” means the same thing as “impossible.”
Eisner’s Running Example

Two paths for (aabb, xz):

Path 1:
0 \(\xrightarrow{a:x/.63} 0 \xrightarrow{a:ε/.07} 1 \xrightarrow{b:ε/.03} 2 \xrightarrow{b:z/.4} 2/.5\)
Path score = 0.0002646

Path 2:
0 \(\xrightarrow{a:x/.63} 0 \xrightarrow{a:ε/.07} 1 \xrightarrow{b:z/.12} 2 \xrightarrow{b:ε/.1} 2/.5\)
Path score = 0.0002646

Score of (aabb, xz) = 0.0005296
Weighted Relations and Probabilities

• Let $f(x, y)$ be the function corresponding to a WFST’s assignment of a score to the (input, output) pair $(x, y)$.
  – If $f$ sums to one over all $(x, y)$, then it is a joint distribution.
  – If $f$ sums to one over all $y$, for each $x$, then it is a conditional distribution.
Parameterizing the WFST

• Option 1: every transition or stop gets a parameter.
  – Option 1A: make sure competing choices (transitions from q and stopping in q) sum to 1.

[Diagram of a WFST with labeled transitions and states, indicating 13 free parameters]
Parameterizing the WFST

- Suppose our WFST was built by **composing** two simpler WFSTs.
WFST Composition

\[ \Sigma = \{a, b\} \]
\[ \Delta = \{p, q\} \]

\[ \Sigma = \{p, q\} \]
\[ \Delta = \{x, z\} \]
WFST Composition

Input string in \{a, b\}*

WFST 1

Intermediate string in \{p, q\}*

WFST 2

Output string in \{x, z\}*
WFST Composition

input string in \{a, b\}*

WFST 1

intermediate string in \{p, q\}*

WFST 2

output string in \{x, z\}*
WFST Composition

input string in \{a, b\}*

output string in \{x, z\}*
WFST Composition

• Let \( f \) and \( g \) define the weighted relations for two WFSTs such that \( f \)'s output alphabet and \( g \)'s input alphabet are the same.

Then:
\[
f \circ g (x, z) = \sum_y f(x, y) \times g(y, z)
\]

– Either \( f \) or \( g \) or both can be a set (instead of a relation).
– Either \( f \) or \( g \) can be unweighted (scores are 0 or 1).
– If both are unweighted sets (FSAs) then this is \textbf{intersection}.

• If \( f \) is a joint distribution \( p(x, y) \) and \( g \) is a conditional distribution \( p(z \mid y) \), we now have a probabilistic model over three string random variables.
WFST Composition

(4,7) self-loop is really $a \rightarrow p \rightarrow \epsilon$

(4,6) self-loop is really $a \rightarrow p \rightarrow x$

(5,7) self-loops are really $b \rightarrow p \rightarrow \epsilon$ and $b \rightarrow q \rightarrow z$

(5,6) self-loop is really $b \rightarrow \{p, q\} \rightarrow x$
WFST Composition

6 + 1 = 7 free parameters
Aside 1

• Eisner suggests another way to write down weighted regular relations, as probabilistic regular expressions.
  – Build up from atomic expressions $a:b$, with $a$ in $\Sigma^*$ and $b$ in $\Delta^*$
  – Concatenation, probabilistic union, probabilistic closure.

• Almost no work on this in NLP or ML, as far as I’ve seen.
Noisy Channel Model

Source WFSA → idealized output of predictor

Channel WFST

observable input to predictor
Noisy Channel Model

Source WFSA

→

idealized output of predictor

Channel WFST

→

observable input to predictor
Historical Note

• *Unweighted* FSTs were developed largely for designing and implementing models of the *morphology* of natural languages.
  – Huge amount of work at Xerox.
  – Also used in information extraction.

• Very useful for hand-construction of morphological rules individually, then assemble them by concatenation, union, composition, etc.
Parameterizations

1. Every arc gets one probability
2. Every “original” arc gets one probability
3. Log-linear distribution with shared features all over the WFST
   – This is really the most general, since features could be identities of arcs or of “original” arcs!
Exercise

• How to represent an HMM as a WFST?
  – MEMM?
  – Chain-structured CRF?

• How to represent stochastic edit distance as a WFST?
  – Elegant way to design a wide range of edit operations: composition of WFSTs
Back to Learning

• We want a *general* method learning the parameters from data, even when all layers are not known.
  – EM for HMMs is well known (Baum, 1972)
  – EM for stochastic edit distances is well known (Ristad and Yianilos, 1996)

• If our WFST was constructed by composing simpler machines, we might want to keep the original parameterization.
  – I.e., learn weights for individual machines jointly.
Very General Formulation of Learning

• Flexibility in parameterization, including
  – one probability per arc
  – one probability per “original” arc
  – log-linear distribution over arcs from a state, with parameter tying throughout the WFST

• Learn from samples of (input, output) pairs where \( x_i \subseteq \Sigma^* \) and \( y_i \subseteq \Delta^* \) (paths not observed).
  – supervised: each \( x_i \) and each \( y_i \) is a single string
  – unsupervised: \( y_i = \Delta^* \)
  – lots of possibilities in between
Maximum Likelihood Estimation

\[
\max_{\theta} \prod_{i} f_{\theta}(x_i, y_i) = \max_{\theta} \prod_{i} \sum_{a \in \text{Paths}(x_i, y_i)} \prod_{j=1}^{|a|} \text{weight}_{\theta}(a_j)
\]

- You can view this as a generalization of Baum-Welch training or EM for stochastic edit distances.
- Each example’s total score is a **path sum** over the scores of paths that
  - are allowed by the WFST
  - match the input set \(x_i\)
  - match the output set \(y_i\)
- Recall that \(f\) might be a joint or conditional probability distribution and \(\theta\) might be log-linear or multinomial parameters.
Expectation Maximization

• E step (one example): find the distribution over paths given $x_i$ and $y_i$:

$$\sum_{a \in \text{Paths}(x_i, y_i)} \prod_{j=1}^{|a|} \text{weight}_{\theta}(a_j)$$

• M step: update $\theta$ to make those paths more likely (exact form depends on parameterization).

$$\max_{\theta} \sum_a E_{\theta(t-1)}[\text{freq}(a)] \log \text{weight}_{\theta}(a)$$
M Step

$$\max_{\theta} \sum_a \mathbb{E}_{\theta^{(t-1)}}[freq(a)] \log \text{weight}_{\theta}(a)$$

- Parameterization 1 (one probability per arc):
  $$\max_{\theta} \sum_a \mathbb{E}_{\theta^{(t-1)}}[freq(a)] \log \theta_a$$

- Parameterization 2 (one probability per original arc): “unwind” $p(a)$ into product of original arc probabilities.
“Unwinding” Example

(4, 6) self-loop is really $a \rightarrow p \rightarrow x$

0’s self-loop with $a:x$ is really

4’s self-loop with $a:p$ and

6’s self-loop with $p:x$
M Step

\[
\max_{\theta} \sum_a \mathbb{E}_{\theta^{(t-1)}}[\text{freq}(a)] \log \text{weight}_{\theta}(a)
\]

• Parameterization 1 (one probability per arc):

\[
\max_{\theta} \sum_a \mathbb{E}_{\theta^{(t-1)}}[\text{freq}(a)] \log \theta_a
\]

• Parameterization 2 (one probability per original arc): “unwind” p(a) into product of original arc probabilities.

• Parameterization 3 (log-linear and most general):

\[
\max_{\theta} \sum_a \mathbb{E}_{\theta^{(t-1)}}[\text{freq}(a)] \left( \theta^\top g(a) - \log \sum_{a' \in \text{Competitors}(a)} \exp \theta^\top g(a') \right)
\]
Expectation Maximization

• E step (one example): find the distribution over paths \textit{given} \(x_i\) and \(y_i\):

\[
\sum_{a \in \text{Paths}(x_i, y_i)} \prod_{j=1}^{|a|} \text{weight}_\theta(a_j)
\]

\[
\frac{\sum_{a \in \text{Paths}(x_i, y_i)} |a| \prod_{j=1}^{|a|} \text{weight}_\theta(a_j)}{\sum_{a \in \text{Paths}} |a| \prod_{j=1}^{|a|} \text{weight}_\theta(a_j)}
\]

• M step: update \(\theta\) to make those paths more likely (exact form depends on parameterization).

\[
\max_{\theta} \sum_{a} \mathbb{E}_{\theta(t-1)}[\text{freq}(a)] \log \text{weight}_\theta(a)
\]
E Step

for a generative model, $Paths(\Sigma^*, \Delta^*)$
for a conditional model, $Paths(x_i, \Delta^*)$

• The likelihood value for one example is the ratio of two path sums.
  – The denominator path sum is the same for all examples in the generative case.
• But the E step’s real job is to calculate sufficient statistics that the M step needs!
Best Path

• General idea: take \( \mathbf{x} \) and build a graph.
• Score of a path factors into the edges.

\[
\arg \max_y \mathbf{w}^T \mathbf{g}(\mathbf{x}, y) = \arg \max_y \mathbf{w}^T \sum_{e \in \text{Edges}} f(e) \mathbf{1}\{e \text{ is crossed by } y\text{'s path}\}
\]

• Decoding is finding the best path.

The Viterbi algorithm is an instance of finding a best path!
Best Path \textbf{Sum}

- General idea: take $\mathbf{x}$ and build a graph.
- Score of a path factors into the edges.

$$
\log \sum_y \exp \mathbf{w}^T g(\mathbf{x}, \mathbf{y}) = \log \sum_y \exp \mathbf{w}^T \sum_{e \in \text{Edges}} f(e) 1\{e \text{ is crossed by } \mathbf{y}'\text{s path}\}
$$

- Decoding is finding the best path \textbf{sum}

The Viterbi algorithm is an instance of finding a best path \textbf{sum}!
Expected Feature Counts

• The E step’s real job, in the most general case, is to calculate the expected feature counts in the examples, under the current model.

• We’ve seen this before!
  – Forward-backward; you can do it that way
  – Eisner suggests a different way, where the usual “plus-times” semiring is extended and the expectations are obtained in a single pass.
Semirings

The tuple \((K, \oplus, \otimes, 0, 1)\) is a **semiring** if:

- \(K\) is a set of values
- \(\oplus\) is a commutative, binary operation \(K \times K \to K\) with identity element \(0\)
- \(\otimes\) is a binary operation \(K \times K \to K\) with identity element \(1\)
  - for composition, \(\otimes\) must be commutative
- \(\otimes\) distributes over \(\oplus\): \(a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)\).
- \(0\) annihilates \(K\): \(a \otimes 0 = 0\)
- to handle infinite sets of cyclic paths, we need a unary closure operator \(*\) such that \(a^* = 1 \oplus a \oplus (a \otimes a) \oplus (a \otimes a \otimes a) \oplus \ldots\)
Semirings You Know

<table>
<thead>
<tr>
<th>interpretation of weights</th>
<th>want to compute</th>
<th>weights</th>
<th>“plus”</th>
<th>“times”</th>
</tr>
</thead>
<tbody>
<tr>
<td>probabilities</td>
<td>p(s)</td>
<td>[0, 1]</td>
<td>+</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>best path prob.</td>
<td>max</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>log-probabilities</td>
<td>log p(s)</td>
<td>(-∞, 0]</td>
<td>log+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>best path log-prob.</td>
<td>max</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>costs</td>
<td>min-cost path</td>
<td>[0, +∞)</td>
<td>min</td>
<td>+</td>
</tr>
<tr>
<td>Boolean</td>
<td>s in language?</td>
<td>{0, 1}</td>
<td>V</td>
<td>∧</td>
</tr>
<tr>
<td>strings</td>
<td>s itself</td>
<td>Σ*</td>
<td>set-union</td>
<td>concat.</td>
</tr>
</tbody>
</table>
The Expectation Semiring

• Instead of a weight storing just a “forward” score or probability, also store a value in V.
  – For us, V is vectors of (expected) feature counts.
  – Assign to each arc a value corresponding to its local feature vector.

• Define operations:

\[(p, v) \otimes (p', v') = (pp', pv' + p'v)\]
\[(p, v) \oplus (p', v') = (p + p', v' + v')\]
\[(p, v)^* = (p^*, p^*vp^*)\]

• Result: final value contains path sum and feature expectations.
What’s Really Happening?

• We are manipulating **weighted relations**, not WFSTs.

• The expectation semiring’s values are scores and **gradients** of scores with respect to $\theta$
  – Forward-backward (e.g., for CRFs) is doing the same thing, only using the chain rule for derivatives to define a second pass (the backward pass).
  – The expectation semiring lets you avoid the backward pass and the per-arc products of forward and backward probabilities.
  – But it’s probably slower in practice.
Other Goodies
in the Paper and Later Work

• Analysis as the “algebraic path” problem, links to a range of speedups (e.g., for acyclic graphs).

• Viterbi variant of the expectation semiring.

• Probabilistic regular expressions idea.
  – Potential for rapid incorporation of expert intuitions into data-driven systems?

• Li and Eisner (2009) goes to second-order expectation semirings!

• Dreyer and Eisner (2009) uses WFSTs to define factors in graphical models!
Closing Notes on Learning

• Eisner’s approach is for MLE (and MAP), but his algorithms are actually inference methods for WFSTs.
  – By changing to maximization and incorporating costs, you can do perceptron, structured SVM, and other error-driven learning.

• We didn’t talk at all about learning the structure of finite-state models!
  – There’s a rich formal literature on this, and not too many papers that attempt it for real problems.
  – I gave one classic citation on the wiki (Stolcke and Omohundro, 1993).
Toolkits

- FSM libraries (AT&T)
  - Free binaries
  - Implements pretty much everything you need to build weighted and unweighted FS recognizers and transducers ... except training!

- Xerox FS toolkit
  - Web demo; software can be purchased
  - No weights

- RWTH FSA toolkit
  - Newer, open-source
  - Not sure what’s implemented

- OpenFST (Google)
  - New incarnation of FSM libraries
  - Free and open source!
# Notes on Algorithms

<table>
<thead>
<tr>
<th>epsilon-removal</th>
<th>Mohri 2002</th>
<th>limitations</th>
<th>polytime</th>
</tr>
</thead>
<tbody>
<tr>
<td>determinize</td>
<td>Mohri 1997</td>
<td>not all transducers</td>
<td>✓</td>
</tr>
<tr>
<td>minimize</td>
<td>Eisner 2003</td>
<td>not all semirings</td>
<td>✓</td>
</tr>
</tbody>
</table>
Conclusion

• WFSTs are extremely general and powerful
  – People use them to implement or approximate almost everything in NLP, IE, and MT
• You should know this abstraction, even if you don’t use it every day.