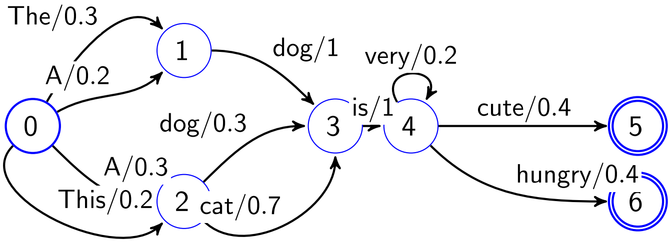
Lecture 17: Practical WFSTs

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2020

Outline

Weighted Finite State Acceptors



- An **FSA** specifies a set of strings. A string is in the set if it corresponds to a valid path from start to end, and not otherwise.
- A **WFSA** also specifies a probability mass function over the set.

Semirings

A **semiring** is a set of numbers, over which it's possible to define a operators $\otimes$ and $\oplus$, and identity elements $\bar{1}$ and $\bar{0}$.

- The **Probability Semiring** is the set of non-negative real numbers $\mathbb{R}_+$, with $\otimes = \cdot$, $\oplus = +$, $\bar{1} = 1$, and $\bar{0} = 0$.
- The **Log Semiring** is the extended reals $\mathbb{R} \cup \{\infty\}$, with $\otimes = +$, $\oplus = -\text{logsumexp}(-,-)$, $\bar{1} = 0$, and $\bar{0} = \infty$.
- The **Tropical Semiring** is just the log semiring, but with $\oplus = \min$. In other words, instead of adding the probabilities of two paths, we choose the best path:

$$a \oplus b = \min(a, b)$$

Mohri et al. (2001) formalize it like this: a **semiring** is $K = \{\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1}\}$ where $\mathbb{K}$ is a set of numbers.

Best-Path Algorithm for a WFSA

- Input string, $S = [s_1, \ldots, s_K]$. For example, the string "A dog is very very hungry" has $K = 5$ words.
- Transitions, t, each have predecessor state $p[t] \in Q$, next state $n[t] \in Q$, weight $w[t] \in \overline{\mathbb{R}}$ and label $\ell[t] \in \Sigma$.
- **Initialize** with path cost either $\bar{1}$ or $\bar{0}$:

$$\delta_0(i) = \begin{cases} \bar{1} & i = \text{initial state} \\ \bar{0} & \text{otherwise} \end{cases}$$

- **Iterate** by choosing best incoming transition:

$$\delta_k(j) = \underset{t:n[t]=j,\ell[t]=s_k}{\text{best}} \delta_{k-1}(p[t]) \otimes w[t]$$

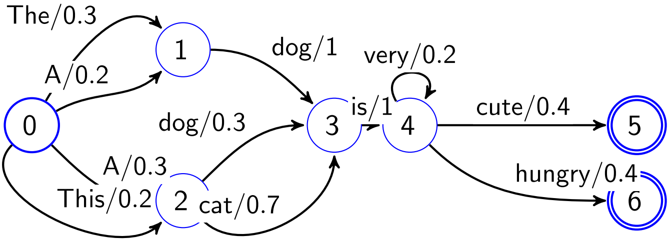
$$\psi_k(j) = \underset{t:n[t]=j,\ell[t]=s_k}{\text{argbest}} \delta_{k-1}(p[t]) \otimes w[t]$$

- **Backtrace** by reading best transition from the backpointer:

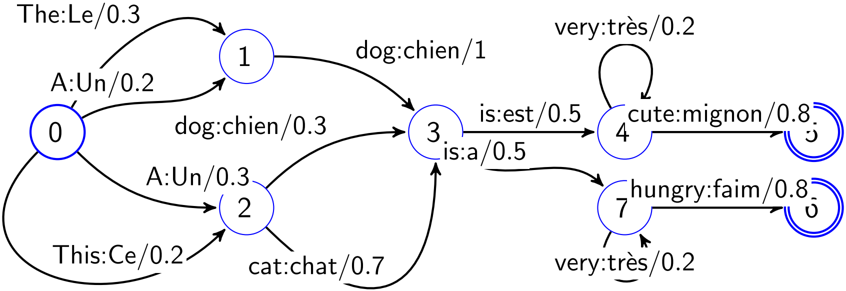
$$t_k^* = \psi(q_{k+1}^*), \qquad q_k^* = p[t_k^*]$$

Determinization

A WFSA is said to be **deterministic** if, for any given (predecessor state $p[e]$, label $\ell[e]$), there is at most one such edge. For example, this WFSA is not deterministic.



Weighted Finite State Transducers



A **(Weighted) Finite State Transducer (WFST)** is a (W)FSA with two labels on every transition:

- An input label, $i[t] \in \Sigma$, and
- An output label, $o[t] \in \Omega$.

The WFST Composition Algorithm

$$C = A \circ B$$

- **States:** The states of C are $Q_C = Q_A \times Q_B$, i.e., $q_C = (q_A, q_B)$.
- **Initial States:** $i_C = (i_A, i_B)$
- **Final States:** $F_C = F_A \times F_B$
- **Input Alphabet:** $\Sigma_C = \Sigma_A$
- **Output Alphabet:** $\Omega_C = \Omega_B$
- **Transitions:**
  1. Every pair $q_A \in Q_A, t_B \in E_B$ with $i[t_B] = \epsilon$ creates a transition t_C from $(q_A, p[t_B])$ to $(q_A, n[t_B])$.
  2. Every pair $t_A \in E_A, q_B \in Q_B$ with $o[t_A] = \epsilon$ creates a transition t_C from $(p[t_A], q_B)$ to $(n[t_A], q_B)$.
  3. Every pair $t_A \in E_A, t_B \in E_B$ with $o[t_A] = i[t_B]$ creates a transition t_C from $(p[t_A], p[t_B])$ to $(n[t_A], n[t_B])$.

Outline

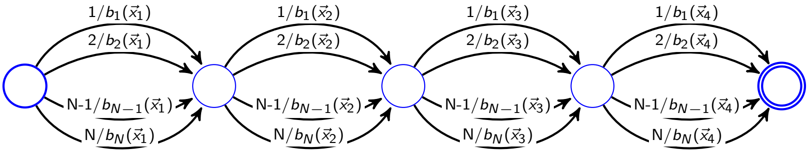
The Standard FSTs in Automatic Speech Recognition

1. The observation, O
2. The hidden Markov model, H
3. The context, C
4. The lexicon, L
5. The grammar, G

MP5 will use L and G, so those are the ones you need to pay attention to. At the input we'll use a transcription T which is basically $T = O \circ H \circ C$, so you won't need to remember the details of those transducers, just their output.

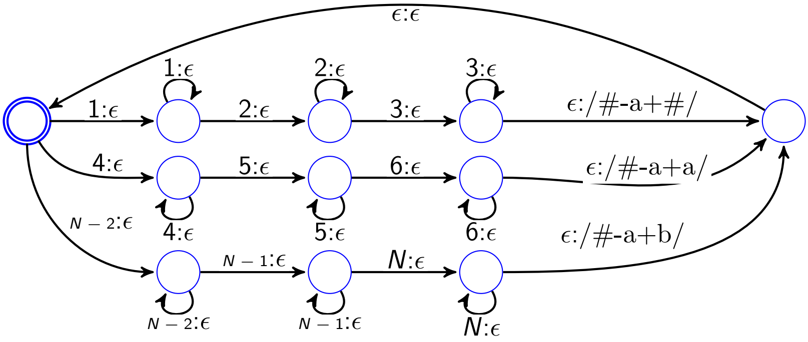
The observation, O

- WFST-based speech recognition begins by turning the speech spectrogram into a WFST.
- The input alphabet is $\Sigma =$the set of acoustic feature vectors.
- The output alphabet is $\Omega = \{1, \ldots, N\}$, the PDFIDs.



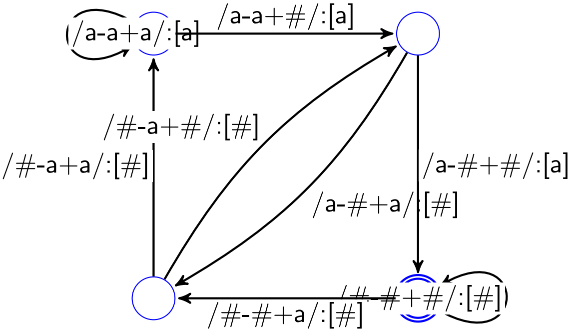
The hidden Markov model, H

- Input alphabet is $\Sigma = \{1, \ldots, N\}$, the set of PDFIDs.
- Output alphabet, Ω, is a set of **context-dependent phone labels**, e.g., **triphones:** $o[t] =$/#-a+b/ means the sound an /a/ makes when preceded by silence, and followed by /b/.



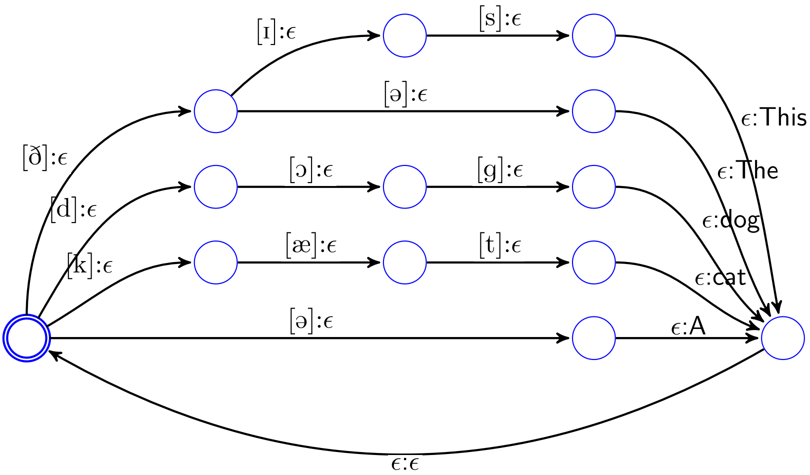
The Context Transducer, C

- Input alphabet, Σ, is **context-dependent phone labels**, e.g., $o[t]$ =/#-a+#/.
- Output alphabet, Ω, is **context-independent phone labels**, e.g., /a/.



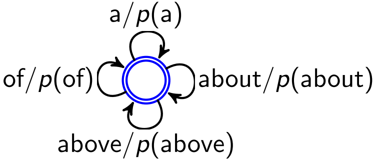
The Lexicon, L

- Input alphabet, Σ, is **phone labels**, e.g., /ə/.
- Output alphabet, Ω, is **words**.



The Grammar, G

- Input alphabet, Σ, is **words**, and
- Output alphabet, Ω, is also **words**.
- Edge weights show $p(w)$



The Standard WFSTs

- H, C, L and G all start in state 0, and end in state 0. That way they can make as many complete loops as necessary.
- O starts at the beginning of the speech file, and ends at the end, with NO LOOPS.
- The most important edge weights are in O and G, the **acoustic model** and **language model** respectively.
- The other transducers (H, C, and L) are used to scale up from 10ms (scale of x_t) to 400ms (scale of w)

Outline

- You already know how to train the acoustic model.
- How can you train the language model?

N-Gram Language Model

An N-gram language model is a model in which the probability of word w_N depends on the $N-1$ words that went before it:

$$p(w_N|\text{context}) \equiv p(w_N|w_1, w_2, \ldots, w_{N-1})$$

Maximum Likelihood N-Grams

Suppose you have some training texts, for example:

Example Training Texts

when telling of nicholas the second the temptation is to start at the dramatic end the july nineteen eighteen massacre of him his entire family his household help and personal physician by which the triumphant communist movement introduced its rule

Maximum Likelihood N-Grams

The **maximum-likelihood** estimates of $p(w_3|w_1, w_2)$ are defined to be the estimates that maximize the **likelihood** of the training data,

$$\mathcal{L} = \prod_{w_i \in \text{training text}} p(w_i|w_{i-2}, w_{i-1}),$$

subject to the constraints that

$$\sum_{w_i} p(w_i|w_{i-2}, w_{i-1}) = 1, \quad p(w_i|w_{i-2}, w_{i-1}) \geq 0$$

Maximum Likelihood N-Grams

The maximum-likelihood estimate turns out to be

$$p(w_i | w_{i-2}, w_{i-1}) = \frac{\#\text{ times } w_i \text{ followed } w_{i-2}, w_{i-1}}{\#\text{ times } w_{i-2}, w_{i-1} \text{ appeared in sequence}}$$

Maximum Likelihood N-Grams: Example

In the following text, the bigram probabilities are

$$p(w_i|w_{i-1} = \text{the}) = \begin{cases} 0.2 & w_i \in \left\{ \begin{array}{c} \text{second} \\ \text{temptation} \\ \text{dramatic} \\ \text{july} \\ \text{triumphant} \end{array} \right\} \\ 0 & \text{otherwise} \end{cases}$$

Example Training Texts

when telling of nicholas the second the temptation is to start at the dramatic end the july nineteen eighteen massacre of him his entire family his household help and personal physician by which the triumphant communist movement introduced its rule

The Problem with Maximum Likelihood

The problem with maximum likelihood is those zeros. For example, suppose you used this model:

$$p(w_i|w_{i-1} = \text{the}) = \begin{cases} 0.2 & w_i \in \left\{ \begin{array}{c} \text{second} \\ \text{temptation} \\ \text{dramatic} \\ \text{july} \\ \text{triumphant} \end{array} \right\} \\ 0 & \text{otherwise} \end{cases}$$

but what the person actually said was:

where is the cafeteria?

Laplace Smoothing

- Laplace proposed the following solution:
- Pretend that every word in the vocabulary has occurred at least once in every possible context.

This results in the following formula:

$$p(w_i|w_{i-2},w_{i-1}) = \frac{1 + \#\text{ times } w_i \text{ followed } w_{i-2},w_{i-1}}{V + \#\text{ times } w_{i-2},w_{i-1} \text{ appeared in sequence}}$$

where V is the number of distinct words in the vocabulary.

Outline

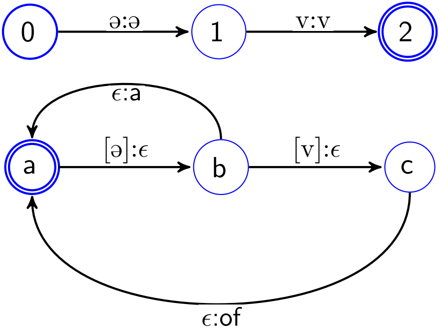
The WFST Composition Algorithm

$$C = A \circ B$$

- **States:** The states of C are $Q_C = Q_A \times Q_B$, i.e., $q_C = (q_A, q_B)$.
- **Initial States:** $i_C = (i_A, i_B)$
- **Final States:** $F_C = F_A \times F_B$
- **Input Alphabet:** $\Sigma_C = \Sigma_A$
- **Output Alphabet:** $\Omega_C = \Omega_B$
- **Transitions:**
  1. Every pair $q_A \in Q_A, t_B \in E_B$ with $i[t_B] = \epsilon$ creates a transition t_C from $(q_A, p[t_B])$ to $(q_A, n[t_B])$.
  2. Every pair $t_A \in E_A, q_B \in Q_B$ with $o[t_A] = \epsilon$ creates a transition t_C from $(p[t_A], q_B)$ to $(n[t_A], q_B)$.
  3. Every pair $t_A \in E_A, t_B \in E_B$ with $o[t_A] = i[t_B]$ creates a transition t_C from $(p[t_A], p[t_B])$ to $(n[t_A], n[t_B])$.

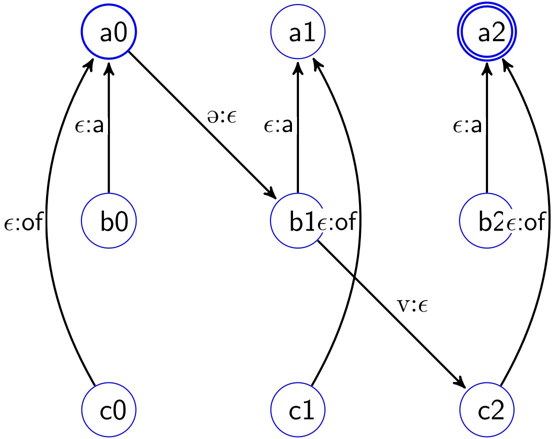
Composition Example

For example, suppose we try to compose this two-phoneme observation with this two-word lexicon:



Composition Example

We wind up with the following transducer:



WFST Composition: Comments

- The ϵ strings add a lot of transitions that are not connected to anything!
- This is necessary, in order to make sure we get the ϵ transition that we actually need.
- The only way to keep the connected transition, and eliminate unconnected ones, is by using a search algorithm to find all the paths through the graph.
- I recommend: do composition **first**, then implement the search algorithm as part of **topological sorting**.

Outline

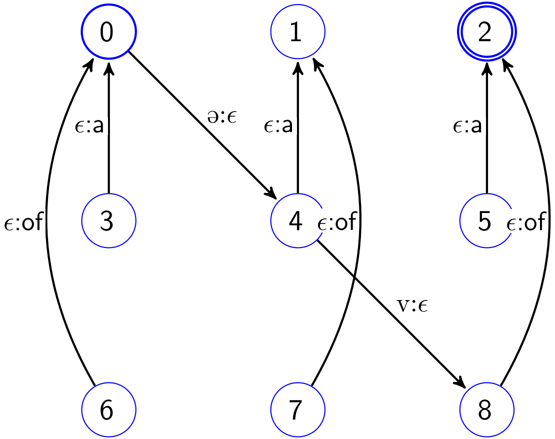
Topological Sorting

A graph is **topologically sorted** if every transition's end state has a higher number than its start state:

$$n[t] \geq p[t] \quad \forall t$$

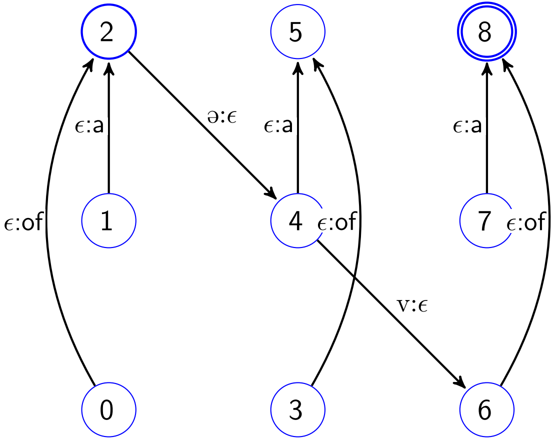
Topological Sorting: Example

This graph is not topologically sorted:



Topological Sorting: Example

This graph **is** topologically sorted:



Why Topologically Sort?

The following algorithms are all **much** more efficient if a graph is first topologically sorted:

- best-path
- forward algorithm
- backward algorithm

Why **Not** Topologicaly Sort?

- A graph with cycles cannot be topologically sorted.
- If your code doesn't use an **explored** set, you'll wind up in an infinite loop.
- If your code uses an **explored** set, after finishing your topological sort, the graph will still not be topologically sorted (because there is no topological sort).

Topological Sort Algorithm = Breadth-First Search Algorithm = Dijkstra's Algorithm

- Input: WFST A.
- Output: WFST B, a copy of A with topologically sorted states, and with unconnected paths removed.
- Required data structures:
  1. A queue called the **frontier**
  2. A set called the **explored** set (optional, but useful).
  3. A dict **A2B**:$Q_A \rightarrow Q_B$.
- Initialization:
  1. Put i_A into the frontier
  2. Create state $i_B = A2B[i_A]$.

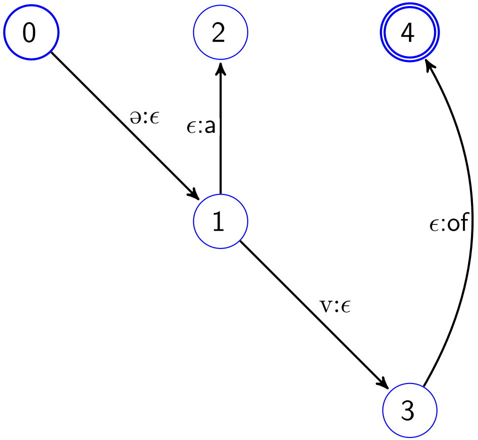
Topological Sort Algorithm = Breadth-First Search (BFS) Algorithm = Dijkstra's Algorithm

While the frontier is not empty:

1. Shift the next state, p_A, off the **frontier**, and put it in the **explored** set.
2. For each transition t_A starting in p_A:
    1. Find its end state n_A.
    2. Look up $p_B = A2B[p_A]$ and $n_B = A2B[n_A]$. If n_B does not exist, create it.
    3. Create a transition t_B from p_B to n_B.
    4. If n_A is not in **frontier** or **explored**, put it in **frontier**.

Topological Sorting: Example

The BFS algorithm topologically sorts, and also eliminates unconnected transitions, so we end up with:



Outline

Best-Path Algorithm for a WFST

Best-path for a WFST is just like for a WFSA, except **we no longer have to worry about the input string!** We assume that you've already composed $O \circ H \circ C \circ L \circ G$ and topologically sorted, so that **all remaining paths in the graph match the input string**. So best-path becomes very simple:

- **Initialize** with path cost either $\bar{1}$ or $\bar{0}$:

$$\delta_0(i) = \begin{cases} \bar{1} & i = \text{initial state} \\ \bar{0} & \text{otherwise} \end{cases}$$

- **Iterate** over states, $j \in Q$:

$$\delta(j) = \underset{t:n[t]=j}{\text{best}} \delta_{k-1}(p[t]) \otimes w[t]$$

$$\psi(j) = \underset{t:n[t]=j}{\text{argbest}} \delta_{k-1}(p[t]) \otimes w[t]$$

- **Backtrace** by reading best transition from the backpointer:

$$t^*(j) = \psi(j), \qquad q^*(t) = p[t^*(j)]$$

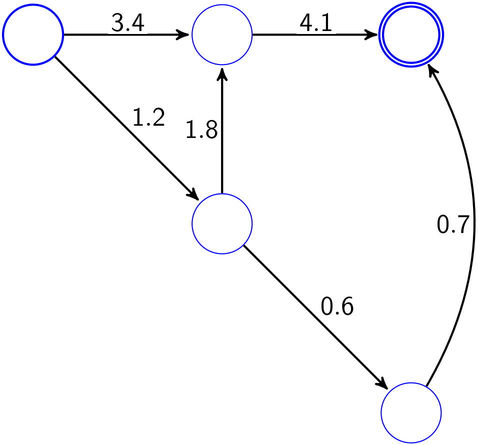
Best-Path Algorithm for a Topologically Sorted WFST

The best-path algorithm is very efficient for a topologically sorted WFST:

1. Sort the transitions in ascending order of their start state.
2. Then step through the transitions in order, checking, for each transition, whether or not $\delta(p[t]) \otimes w[t]$ is better than $\delta(n[t])$. If it is, update $\delta(n[t])$.
3. Topological sort = all transitions for which $j = p[t]$ are sorted after the transitions for which $j = n[t]$.

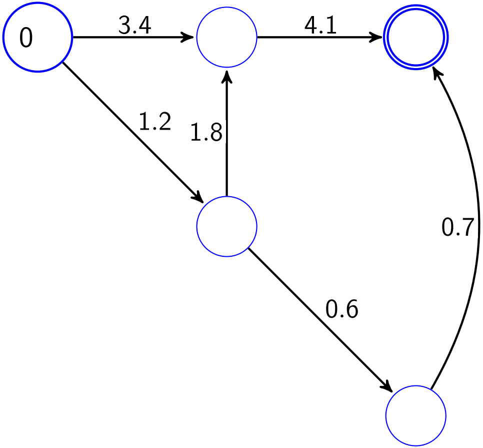
Best-Path Example

Suppose this graph now has these surprisal weights:



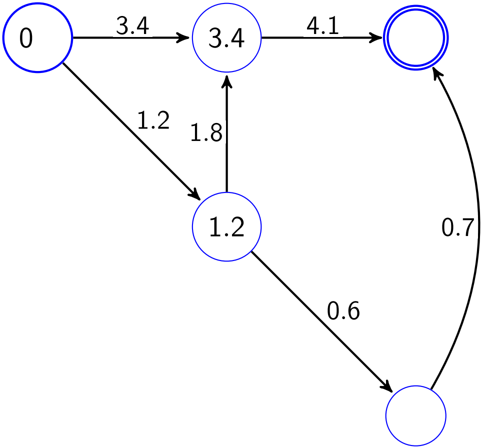
Best-Path Example

Start with $\delta(0) = 0$:



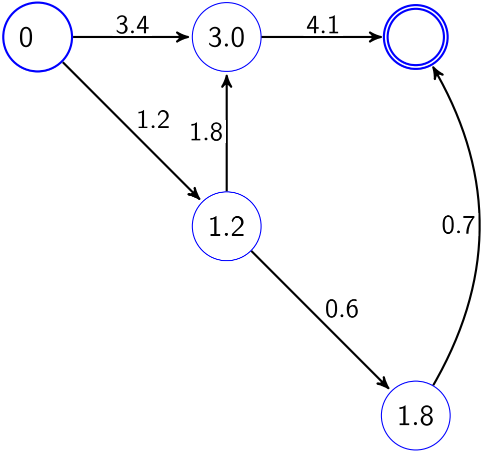
Best-Path Example

Update all the states that can be reached from $q = 0$:



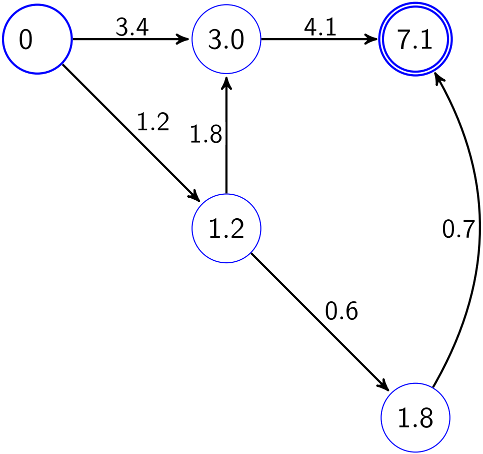
Best-Path Example

Then, states that can be reached from $q = 1$:



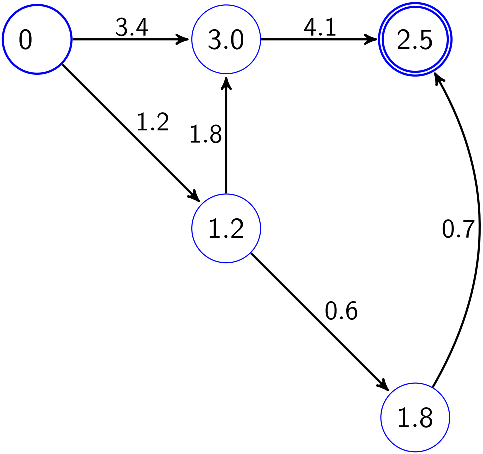
Best-Path Example

Then, states that can be reached from $q = 2$:



Best-Path Example

Then, states that can be reached from $q = 3$:



Outline

What do you re-estimate?

Suppose we want to re-estimate the weight of transition t as the conditional probability of t given its preceding state, $p[t] = j$:

$$w[t] = p(t|p[t])$$

A reasonable way to re-estimate this would be

$$w[t] = \frac{E[\# \text{ times edge } t \text{ was taken}]}{E[\# \text{ times state } p[t] = j \text{ was reached}]}$$

What do you re-estimate?

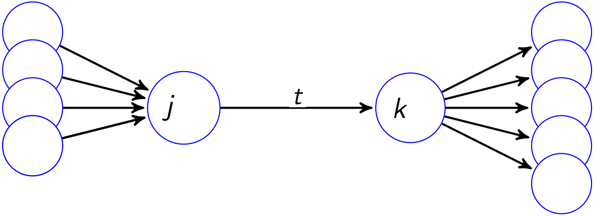
We don't really want to re-estimate edges in the whole stack, $OHCLG = O \circ H \circ C \circ L \circ G$, because O is just one observation file. What we really want is to estimate edges of a particular transducer, e.g., the lexicon.

$$w[t_L] = \frac{E[\#\text{ times } L\text{'s edge } t_L \text{ was taken}]}{E[\#\text{ times } L\text{'s state } p[t_L] = j \text{ was reached}]}$$
$$= \frac{\sum_{t_{OCHLG} \subset t_L} p(t_{OHCLG})}{\sum_{t_L : p[t_L]=j} \sum_{t_{OCHLG} \subset t_L} p(t_{OHCLG})}$$

1. Find the probability of every transition in the full-stack, $p(t_{OHCLG})$,
2. Add over all of the full-stack transitions, t_{OHCLG}, that correspond to lexicon transition t_L (notation: $t_{OHCLG} \subset t_L$).
3. Divide by the marginal.

Next question: how do we find $p(t_{OHCLG})$?

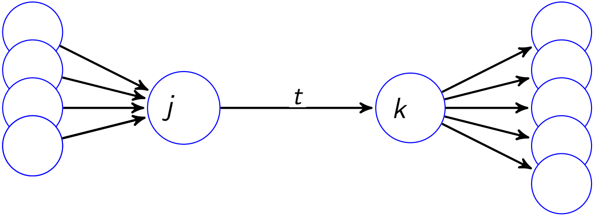
Probability of transition t = Sum of probs of paths including t



Use $\pi = [0, 1, \ldots, j, k, \ldots]$ to mean a path through the whole transducer. It has partial paths $\pi[: j] = [0, 1, \ldots, j]$ and $\pi[: j] = [k, \ldots]$. Then

$$p(t) = \sum_{\pi \text{ includes } t} p(\pi)$$

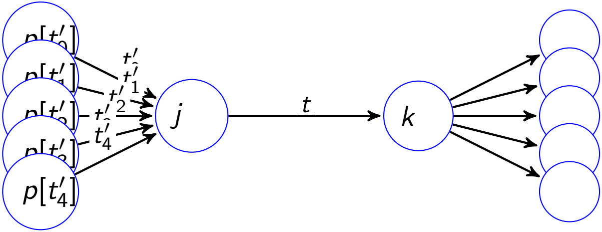
WFST Forward-Backward Algorithm



$$p(t) = \sum_{\pi \text{ includes } t} p(\pi) = \alpha(p[t])w[t]\beta(n[t]),$$

- $\alpha(j) = \sum_{\pi[:j]} p(\pi[:j])$ is the probability of reaching state j.
- $w[t] = p(t|p[t])$ is the probability of taking transition t, given that we reached state $p[t]$.
- $\beta(k) = \sum_{\pi[k:]} p\left(\pi[k:] | k\right)$ is the probability of making it to the end of the WFST, given that we made it to state k.

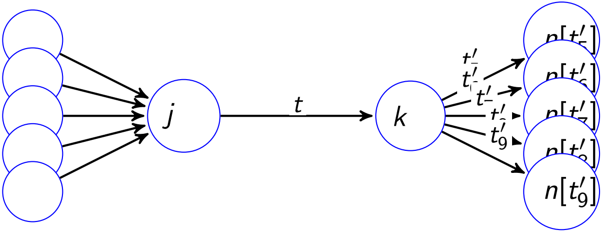
FST Forward Algorithm



First, we need to find $\alpha(j)$:

$$\begin{aligned}\alpha(j) &= \sum_{\pi[:j]} p(\pi[:j]) \\ &= \sum_{t':n[t']=j} \alpha(p[t'])w[t']\end{aligned}$$

FST Backward Algorithm



Then, we need to find $\beta(k)$:

$$
\begin{aligned}
\beta(j) &= \sum_{\pi[k:]} p(\pi[k:]) \\
&= \sum_{t':p[t']=k} w[t']\beta(n[t'])
\end{aligned}
$$

Re-estimation: putting it all back together

Then we just re-estimate the probability of every transition t_L by adding up all the transitions t in OHCLG. If it helps you to remember the idea, we can define a ξ probability, like in HMMs:

$$\xi(t_L) = \sum_{t \subset t_L} \alpha(p[t]) w[t] \beta(n[t])$$

$$w[t_L] = \frac{\xi(t_L)}{\sum_{t':p[t']=p[t_L]} \xi(t')}$$

Outline

The Standard FSTs in Automatic Speech Recognition

1. The observation, O, maps acoustic vectors to PDFIDs
2. The hidden Markov model, H, maps PDFIDs to triphones
3. The context transducer, C, maps triphones to phones
4. The lexicon, L, maps phones to words
5. The grammar, G, computes the probability of a word sequence

MP5 will use L and G, so those are the ones you need to pay attention to.

Laplace Smoothing: Unigram Language Model

- Laplace proposed the following solution:
- Pretend that every word in the vocabulary has occurred at least once.

This results in the following formula:

$$p(w) = \frac{1 + \#\ \text{times}\ w\ \text{occurred}}{V + \#\ \text{word tokens in training data}}$$

where V is the number of distinct words in the vocabulary.

The WFST Composition Algorithm

$$C = A \circ B$$

- **States:** The states of C are $Q_C = Q_A \times Q_B$, i.e., $q_C = (q_A, q_B)$.
- **Initial States:** $i_C = (i_A, i_B)$
- **Final States:** $F_C = F_A \times F_B$
- **Input Alphabet:** $\Sigma_C = \Sigma_A$
- **Output Alphabet:** $\Omega_C = \Omega_B$
- **Transitions:**
  1. Every pair $q_A \in Q_A, t_B \in E_B$ with $i[t_B] = \epsilon$ creates a transition t_C from $(q_A, p[t_B])$ to $(q_A, n[t_B])$.
  2. Every pair $t_A \in E_A, q_B \in Q_B$ with $o[t_A] = \epsilon$ creates a transition t_C from $(p[t_A], q_B)$ to $(n[t_A], q_B)$.
  3. Every pair $t_A \in E_A, t_B \in E_B$ with $o[t_A] = i[t_B]$ creates a transition t_C from $(p[t_A], p[t_B])$ to $(n[t_A], n[t_B])$.

Topological Sort Algorithm = Breadth-First Search (BFS) Algorithm = Dijkstra's Algorithm

While the frontier is not empty:

1. Shift the next state, p_A, off the **frontier**, and put it in the **explored** set.
2. For each transition t_A starting in p_A:
   1. Find its end state n_A.
   2. Look up $p_B = A2B[p_A]$ and $n_B = A2B[n_A]$. If n_B does not exist, create it.
   3. Create a transition t_B from p_B to n_B.
   4. If n_A is not in **frontier** or **explored**, put it in **frontier**.

Best-Path Algorithm for a WFST

Best-path for a WFST is just like for a WFSA, except **we no longer have to worry about the input string!** We assume that you've already composed $O \circ H \circ C \circ L \circ G$ and topologically sorted, so that **all remaining paths in the graph match the input string**. So best-path becomes very simple:

- **Initialize** with path cost either $\bar{1}$ or $\bar{0}$:

$$\delta_0(i) = \begin{cases} \bar{1} & i = \text{initial state} \\ \bar{0} & \text{otherwise} \end{cases}$$

- **Iterate** over states, $j \in Q$:

$$\delta(j) = \underset{t:n[t]=j}{\text{best}} \delta_{k-1}(p[t]) \otimes w[t]$$

$$\psi(j) = \underset{t:n[t]=j}{\text{argbest}} \delta_{k-1}(p[t]) \otimes w[t]$$

- **Backtrace** by reading best transition from the backpointer:

$$t^*(j) = \psi(j), \qquad q^*(t) = p[t^*(j)]$$

Re-estimation

$$\alpha(j) = \sum_{t':n[t']=j} \alpha(p[t'])w[t']$$

$$\beta(j) = \sum_{t':p[t']=k} w[t']\beta(n[t'])$$

$$\xi(t_L) = \sum_{t \subset t_L} \alpha(p[t])w[t]\beta(n[t])$$

$$w[t_L] = \frac{\xi(t_L)}{\sum_{t':p[t']=p[t_L]} \xi(t')}$$