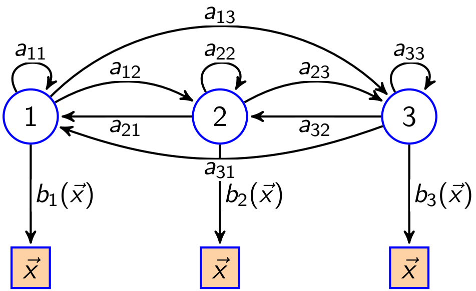
Lecture 15: Weighted Finite State Acceptors (WFSA)

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2020

Outline

The Three Problems for an HMM



- $\pi_i = p(q_1 = i)$ is called the **initial state probability**.
- $a_{ij} = p(q_t = j | q_{t-1} = i)$ is called the **transition probability**.
- $b_j(\vec{x}) = p(\vec{x}_t = \vec{x} | q_t = j)$ is called the **observation probability**.

Recognition: The Forward Algorithm

1. **Initialize:**

$$\alpha_1(i) = \pi_i b_i(\vec{x}_1)$$

2. **Iterate:**

$$\begin{aligned}\alpha_t(j) &= \sum_{i=1}^{N} p(q_{t-1} = i|\vec{x}_1, \ldots, \vec{x}_{t-1}) a_{ij} b_j(\vec{x}_t) \\ &= \sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{ij} b_j(\vec{x}_t)\end{aligned}$$

3. **Terminate:**

$$\ln p(X|\Lambda) = \sum_{j=1}^{N} \alpha_T(j)$$

Segmentation: The Log-Viterbi Algorithm

1. **Initialize:**

$$\ln \delta_1(i) = \ln \pi_i + \ln b_i(\vec{x}_1)$$

2. **Iterate:**

$$\ln \delta_t(j) = \max_{i=1}^{N} \left(\ln \delta_{t-1}(i) + \ln a_{ij} + \ln b_j(\vec{x}_t) \right)$$

$$\psi_t(j) = \underset{i=1}{\overset{N}{\mathrm{argmax}}} \left(\ln \delta_{t-1}(i) + \ln a_{ij} + \ln b_j(\vec{x}_t) \right)$$

3. **Terminate:** Choose the known final state q_T^*.
4. **Backtrace:**

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right)$$

Outline

Finite State Acceptors

All of the material in today's lecture comes from this article:

Article Submitted to Computer Speech and Language

Weighted Finite-State Transducers in Speech Recognition

MEHRYAR MOHRI[1], FERNANDO PEREIRA[2] AND MICHAEL RILEY[1]

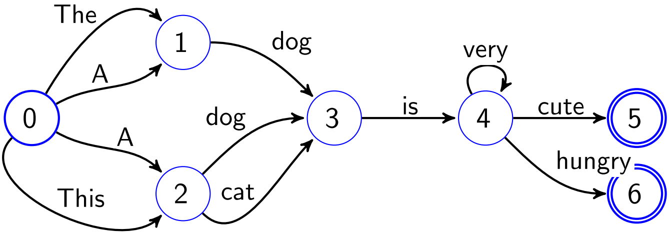
[1] *AT&T Labs – Research*
180 Park Avenue, Florham Park, NJ 07932-0971, USA

[2] *Computer and Information Science Dept., University of Pennsylvania*
558 Moore-GRW, 200 South 33rd Street, Philadelphia, PA 19104 USA

Abstract

We survey the use of weighted finite-state transducers (WFSTs) in speech recognition. We show that WFSTs provide a common and natural representation for HMM models, context-dependency, pronunciation dictionaries, grammars, and alternative recognition outputs. Furthermore, gen-

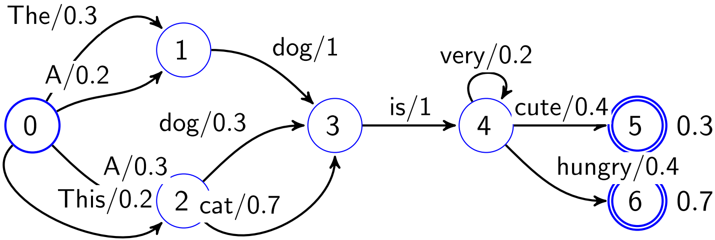
Finite State Acceptors



A **Finite State Acceptor (FSA)**, $A = \{\Sigma, Q, E, i, F\}$, is a finite state machine capable of accepting any string in a (possibly infinite) set.

- Q is a set of states, and E a set of edges.
- Σ is an alphabet of labels that may appear on edges.
- i is the initial state, shown with a thick border. F is the set of final states, shown with doubled borders.

Weighted Finite State Acceptors



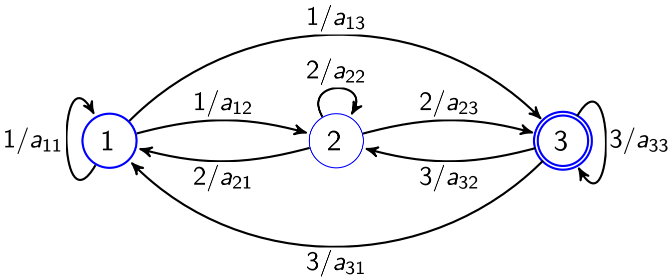
A **Weighted Finite State Acceptor (WFSA)** is an FSA with weights on the edges.

- The edge weights are usually interpreted as conditional probabilities (of the edge given the state), but other interpretations are possible.
- It's also possible to put probabilities on the final states, as shown in this figure (but we don't do this very often).

What it's for

- 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.

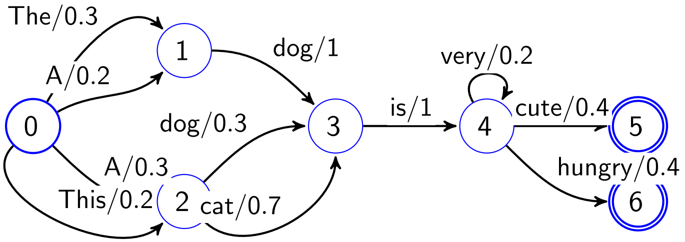
Every Markov Model is a WFSA



A Markov Model (but not an HMM!) may be interpreted as a WFSA: just assign a label to each edge. The label might just be the state number, or it might be something more useful.

Outline

Multiplication: Accumulate on a Path



Multiplication is used to accumulate the weights on a single path through the WFSA. For example, there are two paths matching the sentence "A dog is hungry" Their path weights are

$$p(\text{Path through state 1}) = (0.2)(1)(1)(0.4) = 0.08$$

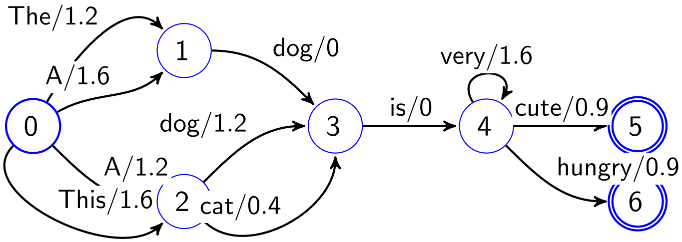
$$p(\text{Path through state 2}) = (0.3)(0.3)(1)(0.4) = 0.036$$

Negative Log Probabilities

WFSAs have floating point underflow problems. The standard solution is to perform all computations using negative log probabilities. Negative log probability ($-\log p(A)$) goes by many names:

- "Surprisal," because you are more surprised if something unlikely happens.
- "Information," because low-probability events are more informative.
- "Cost," because it costs more to take a low-probability path.

WFSA with Negative Log Probabilities



Adding Negative Log Probabilities accumulates the costs on a single path. For example, there are two paths matching the sentence "A dog is hungry" Their path weights are

$$-\ln p(\text{Path through state 1}) = 1.6 + 0 + 0 + 0.9 = 2.5$$

$$-\ln p(\text{Path through state 2}) = 1.2 + 1.2 + 0 + 0.9 = 3.3$$

Otimes Notation

In designing a WFSA, we want our design to be robust, even if we suddenly change between probabilities↔negative log probabilities. Instead of using the standard real-valued "times" operator, and the constants "1" and "0," we use overloaded operators $\otimes$, $\bar{1}$, and $\bar{0}$ whose behavior is determined by the type of their inputs:

- If the inputs are probabilities, then $\otimes$ means "multiply," $\bar{1}$ means "one," and $\bar{0}$ means"zero." Thus, for example

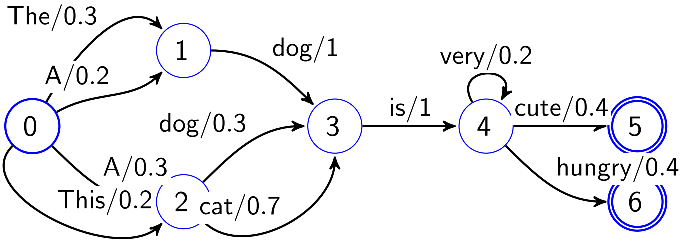
$$(0.2)\otimes(0.7)\otimes\bar{1} = 0.2\cdot 0.7\cdot 1 = 0.14$$
$$(0.2)\otimes\bar{0} = 0.2\cdot 0 = 0$$

- If the inputs are negative log probabilities, then $\otimes$ means "add," $\bar{1}$ means $-\ln(1)=0$, and $\bar{0}$ means $-\ln(0)=\infty$. Thus

$$(1.6)\otimes(0.4)\otimes\bar{1} = 1.6+0.4+0 = 2.0$$
$$(1.6)\otimes\bar{0} = 1.6+\infty = \infty$$

Outline

Finding the Best Path



Often, given an input string, we want to find the **best path** matching that string. This is done using a version of the Viterbi algorithm.

Best-Path Algorithm for a WFSA

Given:

- Input string, $S = [s_1, \ldots, s_T]$. For example, the string "A dog is very very hungry" has $T = 5$ words.
- Edges, e, each have predecessor state $p[e] \in Q$, next state $n[e] \in Q$, weight $w[e] \in \overline{\mathbb{R}}$ and label $\ell[e] \in \Sigma$.
- **Initialize:**

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

- **Iterate:**

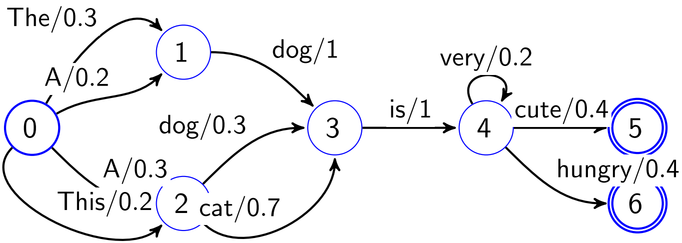
$$\delta_t(j) = \underset{e:n[e]=j,\ell[e]=s_t}{\text{best}} \delta_{t-1}(p[e]) \otimes w[e]$$

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

- **Backtrace:**

$$e_t^* = \psi(q_{t+1}^*), \qquad q_t^* = p[e_t^*]$$

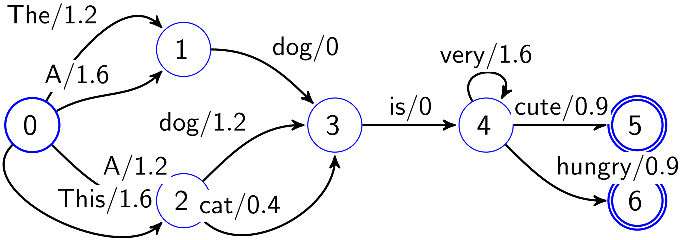
Best Path: Probabilities



After the first two words, "A dog..." we have to compare two possible paths:

$$\delta_2(3) = \text{best}\left(0.2 \otimes 1, 0.3 \otimes 0.3\right) = \text{best}\left(0.2, 0.09\right) = 0.2$$

Best Path: Log Probabilities

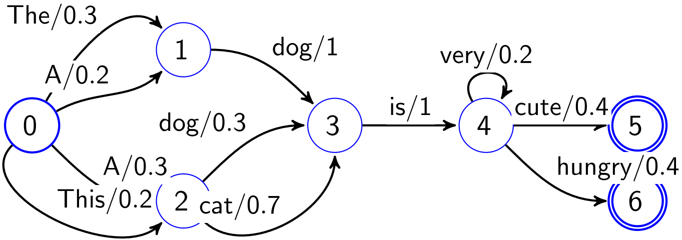


After the first two words, "A dog..." we have to compare two possible paths:

$$\delta_2(3) = \text{best}\,(1.6 \otimes 0, 1.2 \otimes 1.2) = \text{best}\,(1.6, 2.4) = 1.6$$

Outline

Addition: Combine Paths



Addition is used to combine the weights of two different paths. For example, the total probability of the sentence "A dog is hungry" is the sum of the probabilities of its two paths:

$p(\text{A dog is hungry}) = p(\text{Path 1}) + p(\text{Path 2}) = 0.08 + 0.036 = 0.116$

The Oplus Operator

When we convert from probabilities to surprisals, instead of using ordinary (multiplication,addition,1,0), we want to use overloaded operators ($\otimes$,$\oplus$,$\bar{1}$,$\bar{0}$), whose behavior is determined by the type of their inputs:

- If the WFSA is using probability, then $\oplus$ means "addition," and $\bar{0}$ means "zero." Thus, for example

$$(0.08) \oplus (0.06) \oplus \bar{0} = 0.08 + 0.06 + 0 = 0.14$$

- If the WFSA is using negative log probability, then $\oplus$ and $\bar{0}$ should be redefined in some way that gives the desired result. The desired result is that:

$$(-\ln(0.08)) \oplus (-\ln(0.06)) \oplus \bar{0} = -\ln(0.14)$$

The Negative Logsumexp Function

Suppose a and b are negative log probabilities:

$$a = -\ln p(A), \quad b = -\ln p(B)$$

The most computationally efficient way to implement the $\oplus$ operator is also the one that's easiest to understand:

$$a \oplus b = -\ln\left(p(A) + p(B)\right) = -\ln\left(e^{-a} + e^{-b}\right)$$

This function is used so often, in machine learning, that it has a special name. It is called the `logsumexp` function:

$$a \oplus b = -\text{logsumexp}(-a, -b) = -\ln\left(e^{-a} + e^{-b}\right)$$

Logsumexp and Floating Point Underflow

The most computationally efficient way to implement `logsumexp` is also the easiest to understand. It is just:

$$\text{logsumexp}(x, y) = \ln\left(e^x + e^y\right)$$

Unfortunately, that formula may suffer from floating point overflow, e.g., if $x > 100$ or $y > 100$. The following alternative implementation is guaranteed to avoid floating point overflow:

$$m = \max(x, y)$$
$$\text{logsumexp}(x, y) = m + \ln\left(e^{x-m} + e^{y-m}\right)$$

Logsumexp and Max

The following implementation of `logsumexmp` avoids floating point overflow:

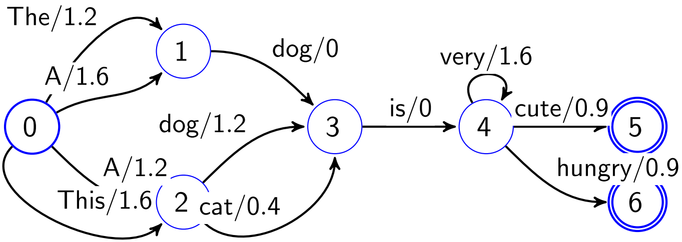
$$m = \max(x, y)$$
$$\text{logsumexp}(x, y) = m + \ln\left(e^{x-m} + e^{y-m}\right)$$

For example, suppose $x > y$, then we get $\text{logsumexp}(x, y) = x + \ln\left(1 + e^{y-x}\right)$. The second term inside the parentheses is $0 \leq e^{y-x} \leq 1$, so

$$\max(x, y) \leq \text{logsumexp}(x, y) \leq \max(x, y) + \ln(2)$$

For this reason, logsumexp is a differentiable approximation of the max operator.

Addition: Combine Paths



Negative Logsumexp is used to combine the surprisals of two different paths. For example, the total surprisal of the sentence "A dog is hungry" is the negative logsumexp of the surprisals of its two paths:

$$p(\text{A dog is hungry}) = (1.6 \otimes 0 \otimes 0 \otimes 0.9) \oplus (1.2 \otimes 1.2 \otimes 0 \otimes 0.9)$$
$$= 2.5 \oplus 3.3 = 2.2$$

$\bar{0}$: The identity element of $\oplus$

The $\oplus$ operator, for surprisal weights, is a negative logsumexp:

$$a \oplus b = -\operatorname{logsumexp}(-a, -b) \leq \min(a, b)$$

The identity element, $\bar{0}$, is the element such that

$$a \oplus \bar{0} = a$$

If you work through the definition of the logsumexp function, you can discover that its identity element is

$$\bar{0} = -\ln(0) = +\infty$$

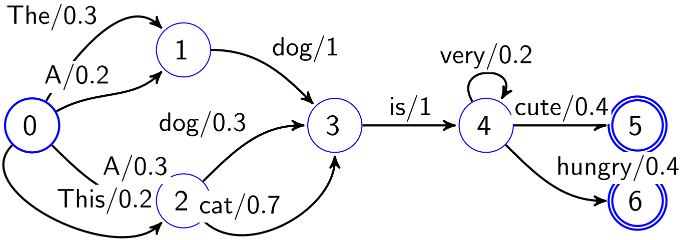
Outline

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.
- If a WFSA is deterministic, then for any given string $S = [s_1, \ldots, s_T]$, there is **at most one** path.
- Determinism makes many other computations very efficient. For example, the best-path algorithm is $\mathcal{O}\{T\}$.

A Non-Deterministic WFSA: Example

This WFSA is not deterministic, because there are two different paths leaving state $p[e] = 0$ that both have the label $\ell[e] =$ "A":



Determinizing a WFSA

Determinizing a WFSA is the creation of a new WFSA such that:

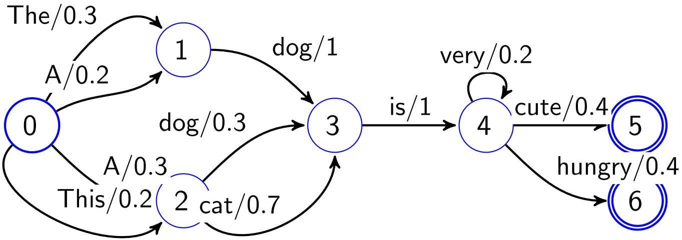
- If A has one or more paths matching any given string, $S = [s_1, \ldots, s_T]$, then A' must have exactly one such path.
- The path weight (probability, surprisal) in A' must be the sum $(\oplus)$ of the weights of all of the paths in A.

How to Determinize a WFSA

The only general algorithm for **determinizing** a WFSA is the following exponential-time algorithm:

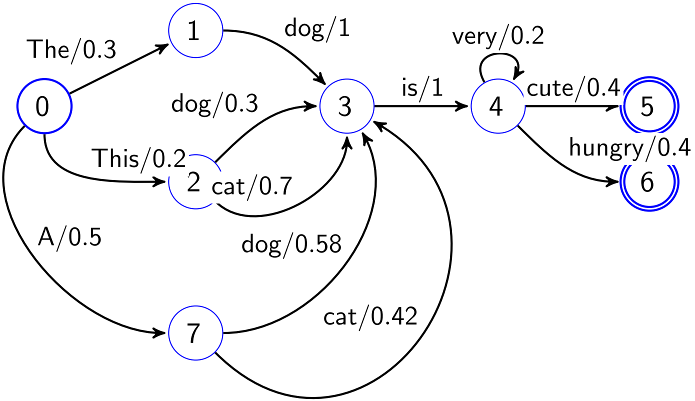
- For every state in A, for every set of edges $e_1, \ldots, e_K$ that all have the same label:
  - Create a new edge, e, with weight $w[e] = w[e_1] \oplus \cdots \oplus w[e_K]$.
  - Create a brand new successor state $n[e]$.
  - For every edge leaving any of the original successor states $n[e_k]$, $1 \leq k \leq K$, whose label is unique:
    - Copy it to $n[e]$, $\otimes$ its weight by $w[e_k]/w[e]$
  - For every set of edges leaving $n[e_k]$ that all have the same label:
    - Recurse!

How to Determinize a WFSA: Example



1. $\oplus$ together the two edges with $\ell[e] =$"A", and create a new state $n[e]$ for them.
2. Copy the successor edge "cat" to the new state.
3. $\oplus$ together the two "dog" successor edges, and copy to the new state.

How to Determinize a WFSA: Example



$$\left(\frac{0.2}{0.5}\right)(1) + \left(\frac{0.3}{0.5}\right)0.3 = 0.58, \qquad \left(\frac{0.3}{0.5}\right)0.7 = 0.42$$

Outline

Summary

- A weighted finite state automaton (WFSA) is a graph (states and edges), each of whose edges carries both a label and a weight.
- The weights may be interpreted as probabilities, or negative log probabilities (surprisals or costs).
- In order to make the math robust to changes between probability$\leftrightarrow$surprisal, we define overloaded operators $\otimes$, $\oplus$, $\bar{1}$, $\bar{0}$, and best whose behavior is determined by the type of their inputs.
- The **best-path** algorithm is just Viterbi, timed according to the input string.
- A **deterministic** WFSA has, for each $(p[e], \ell[e])$ pair, at most one edge.