CSCI 5832
Natural Language Processing

Lecture 12
Jim Martin

Today: 2/13

- **Review Entropy**
- **Back to Parts of Speech**
- **Break**
- **Tagging**

Entropy

- **Defined as**

$$H(S) = -\sum_{w \in V} P(w) \log P(w)$$

Entropy

- **Two views of this…**
  - **As a means of measuring the information (surprise) in a given string with respect to some input**
  - **As a means of measuring how well a given (learned) model fits a given corpus (big long string)**

Cross-Entropy

- Defined as $$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n)$$
- Where p is the true (unknown distribution)
- And m is the distribution defined by the current model
- For any incorrect model H(p) < H(p,m)
  - Why?
  - An incorrect model is in some sense a model that's being surprised by what it sees
  - Surprise means probabilities that are lower than they should be… meaning entropy is higher than it should be.

Entropy

- View the cross-entropy of sequence as the average level of surprise in the sequence
- The more you're surprised by a given corpus, the worse job your model was doing at prediction

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n)$$

So…

- **If you have 2 models m_1 and m_2 the one with the lower cross-entropy is the better one.**

Parts of Speech

- **Start with eight basic categories**
  - **Noun, verb, pronoun, preposition, adjective, adverb, article, conjunction**
- **These categories are based on morphological and distributional properties (not semantics)**
- **Some cases are easy, others are murky**

Parts of Speech

- Two kinds of category
  - Closed class
    - Prepositions, articles, conjunctions, pronouns
  - Open class
    - Nouns, verbs, adjectives, adverbs

Sets of Parts of Speech: Tagsets

- There are various standard tagsets to choose from; some have a lot more tags than others
- The choice of tagset is based on the application
- Accurate tagging can be done with even large tagsets

Tagging

- Part of speech tagging is the process of assigning parts of speech to each word in a sentence… Assume we have
  - A tagset
  - A dictionary that gives you the possible set of tags for each entry
  - A text to be tagged
  - A reason?

Three Methods

- Rules
- Probabilities
- Sort of both

Rules

- Hand-crafted rules for ambiguous words that test the context to make appropriate choices
  - Early attempts fairly error-prone
  - Extremely labor-intensive

Probabilities

- We want the best set of tags for a sequence of words (a sentence)
- W is a sequence of words
- T is a sequence of tags

$$\arg\max P(T \mid W) = \frac{P(W \mid T)P(T)}{P(W)}$$

Probabilities

- We want the best set of tags for a sequence of words (a sentence)
- W is a sequence of words
- T is a sequence of tags

$$\arg\max P(T \mid W) = P(W \mid T)P(T)$$

Tag Sequence: P(T)

- How do we get the probability of a specific tag sequence?
  - Count the number of times a sequence occurs and divide by the number of sequences of that length. Not likely.
  - Make a Markov assumption and use N-grams over tags...
    - P(T) is a product of the probability of N-grams that make it up.

P(T): Bigram Example

- <s> Det Adj Adj Noun </s>
- P(Det|<s>)P(Adj|Det)P(Adj|Adj)P(Noun|Adj)

Counts

- Where do you get the N-gram counts?
- From a large hand-tagged corpus.
  - For N-grams, count all the Tag_i Tag_{i+1} pairs
  - And smooth them to get rid of the zeroes
- Alternatively, you can learn them from an untagged corpus

What about P(W|T)

- First its odd. It is asking the probability of seeing "The big red dog" given "Det Adj Adj Noun"
  - Collect up all the times you see that tag sequence and see how often "The big red dog" shows up. Again not likely to work.

P(W|T)

- We'll make the following assumption (because it's easy)… Each word in the sequence only depends on its corresponding tag. So…

$$P(W \mid T) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

- How do you get the statistics for that?

So…

- **We start with**

$$\arg\max P(T \mid W) = P(W \mid T)P(T)$$

- **And get**

$$\arg\max \prod_{i=2}^{n} P(w_i \mid t_i) * P(t1) * \prod_{i=2}^{n} P(t_i \mid t_{i-1})$$

Break

- **Quiz**
  - **Chapter 2: All**
  - **Chapter 3: Sec 3.1-3.9**
  - **Chapter 4: Sec 4.1-4.3, 4.5, 4.10**
  - **Chapter 5: Pages 1-26**

HMMs

- This is an HMM

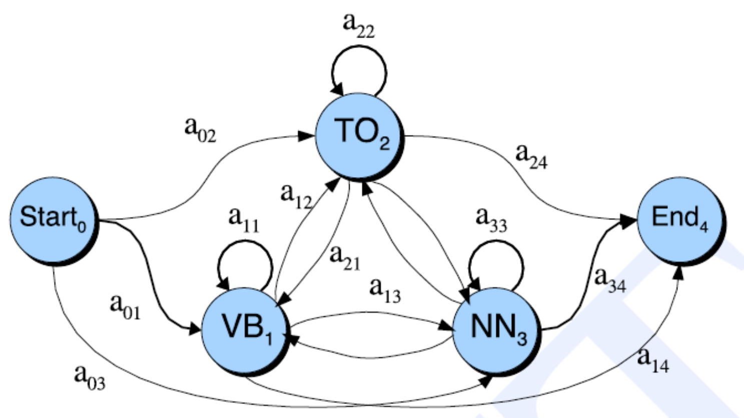
$$\arg\max \prod_{i=2}^{n} P(w_i \mid t_i) * P(t_1) * \prod_{i=2}^{n} P(t_i \mid t_{i-1})$$

- The states in the model are the tags, and the observations are the words.
  - The state to state transitions are driven by the bigram statistics
  - The observed words are based solely on the state that you're in

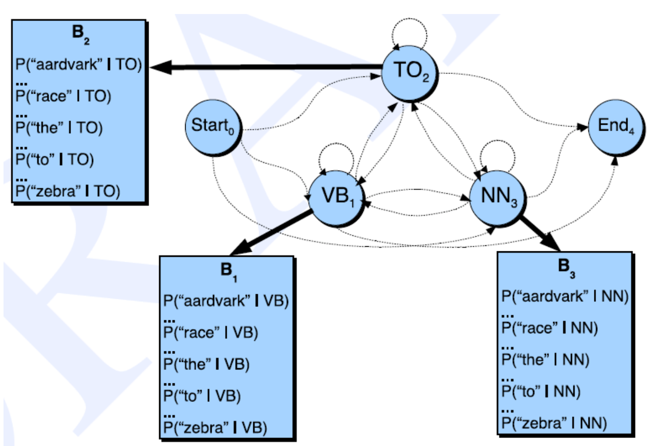
HMMs

- Why hidden?
  - You have a sequence of observations.
  - You have a set of unseen states that gave rise to (generated) that sequence of observations.
    - Those are hidden from you

State Machine



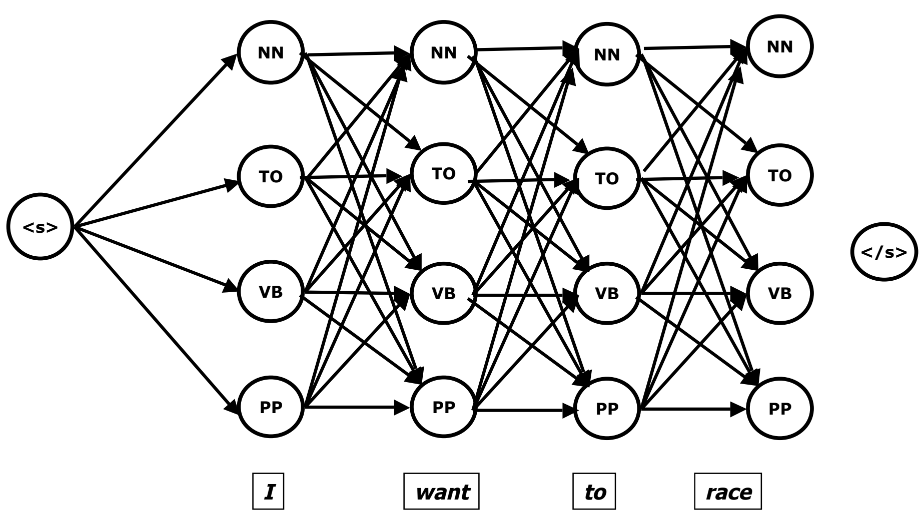
State Machine with Observations

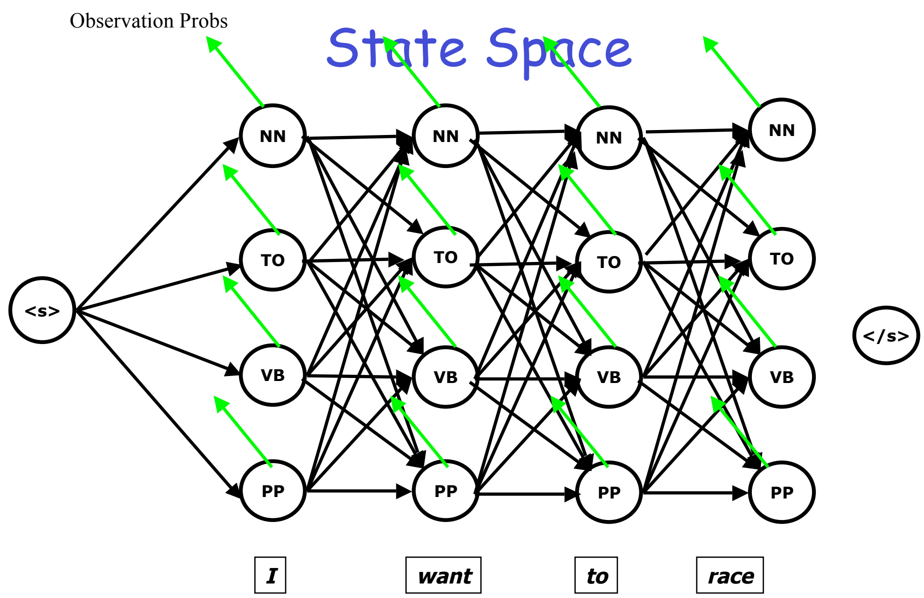


Unwinding in Time

- That diagram isn't too helpful since it really isn't keeping track of where it is in the sequence in the right way.
- So we'll in effect unroll it (since we know the length of the sequence we're dealing with.

State Space





State Space

State Space

- How big is that space?
  - How many paths through the machine?
  - How many things to argmax over?
  - Is that a lot?

State Space

- Fortunately…
  - The markov assumption combined with the laws of probability allow us to use dynamic programming to get around this.

Viterbi Algorithm

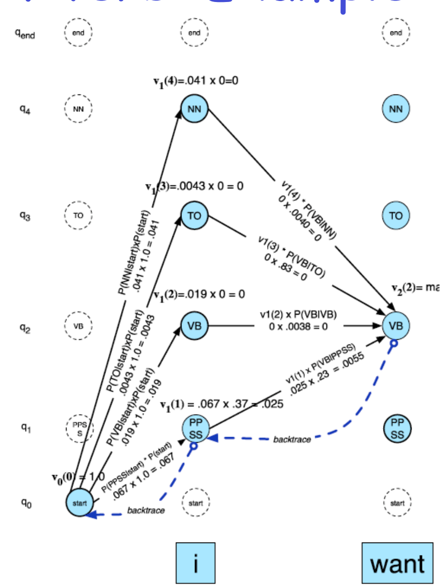
- Efficiently return the most likely path
  - Argmax P(path|observations)
- Sweep through the columns multiplying the probabilities of one row, times the transition probabilities to the previous row, times the appropriate observation probabilities
- And storing the MAX prob at each node
- And keep track of where you're coming from

Viterbi

function VITERBI(*observations* of len *T*, *state-graph*) **returns** *best-path*

num-states ← NUM-OF-STATES(*state-graph*)
Create a path probability matrix *viterbi[num-states+2,T+2]*
viterbi[0,0] ← 1.0
for each time step *t* **from** 1 **to** *T* **do**
  for each state *s* **from** 1 **to** *num-states* **do**
    $viterbi[s,t] \leftarrow \max_{1 \leq s' \leq num\text{-}states} viterbi[s', t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{1 \leq s' \leq num\text{-}states} viterbi[s', t-1] * a_{s',s}$
Backtrace from highest probability state in final column of *viterbi[]* and return path

Viterbi Example



Viterbi

- **How fast is that?**

Next time

- **Quiz on Chapters 2,3,4,and 5**