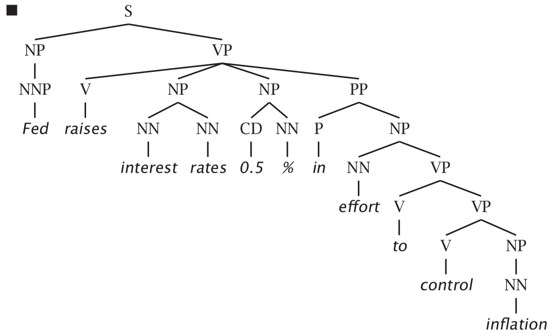
Part of Speech Tagging

FSNLP, chapters 9 and 10

Christopher Manning and Hinrich Schütze

The problem of POS ambiguity

- Structures for: *Fed raises interest rates 0.5% in effort to control inflation* (*NYT* headline 17 May 2000)
-



Part-of-speech ambiguities

		VB					
	VBZ	VBP	VBZ				
NNP	NNS	NN	NNS	CD	NN		
Fed	*raises*	*interest*	*rates*	*0.5*	*%*	*in*	*effort*
						to	*control*
							inflation

The task of part of speech tagging

- A lightweight (usually linear time) processing task, which can usefully empower other applications:
  - Knowing how to pronounce a word: *récord* [noun] vs. *recórd* [verb]
  - Matching small phrasal chunks or particular word class patterns for tasks such as information retrieval, information extraction or terminology acquisition (collocation extraction). Ee.g., just matching nouns, compound nouns, and adjective noun patterns:
    - {A|N}* N
  - POS information can be used to lemmatize a word correctly (i.e., to remove inflections):
    - *saw* [n] → *saw*; *saw* [v] → *see*

The task of part of speech tagging

- ◻ Can differentiate word senses that involve part of speech differences
  - POS can be used as backoff in various class-based models, when too little information is known about a particular word
  - Can be a preprocessor for a parser (often better, but more expensive, to let the parser do the tagging as well)
  - Tagged text helps linguists find interesting syntactic constructions in texts (*ssh* used as a verb)

The linguistics of parts of speech and tag sets

- We're not going to substantively discuss parts of speech in class
  - Read section 3.1 to learn about parts of speech, particularly the kind of richer sets of distinctions commonly made by linguists and in NLP applications
  - Read section 4.3.2 for discussion of POS tag sets used in NLP.
    - There's a handy table explaining tag abbreviations on pp. 141–142

Part of speech tagging

Information sources:

- Sequence of words: syntagmatic information
  - Surprisingly weak information source
  - Many words have various parts of speech – cf. the example above
- Frequency of use of words
  - Surprisingly effective: gets 90+% performance by itself (for English)*
    - This acts as a baseline for performance

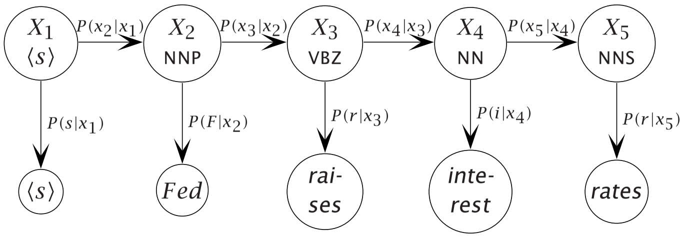
*Even up to 93.7%, based on the results of Toutanova et al. (2003).

(Hidden) Markov model tagger

- View sequence of tags as a Markov chain. Assumptions:
  - **Limited horizon.** $P(X_{i+1} = t^j | X_1, \ldots, X_i) = P(X_{i+1} = t^j | X_i)$
  - **Time invariant (stationary).** $P(X_{i+1} = t^j | X_i) = P(X_2 = t^j | X_1)$

  We assume that a word's tag only depends on the previous tag (limited horizon) and that this dependency does not change over time (time invariance)
- A state (part of speech) generates a word. We assume it depends only on the state

Hidden Markov Models – POS example



- Top row is unobserved states, interpreted as POS tags
- Bottom row is observed output observations
- We normally do supervised training, and then (Bayesian network style) inference to decide POS tags

Standard HMM formalism

- $\langle X, O, \Pi, A, B \rangle$
- X is hidden state sequence; O is observation sequence
- Π is probability of starting in some state (can be folded into A: let $A' = [\Pi | A]$, i.e., $a_{0j} = \pi_j$)
- A is matrix of transition probabilities (top row conditional probability tables (CPTs))
- B is matrix of output probabilities (vertical CPTs)

HMM is also a probabilistic (nondeterministic) finite state automaton, with probabilistic outputs (from vertices, not arcs, in the simplest case)

Probabilistic inference in an HMM

Three fundamental questions:

- Given an observation sequence, compute the most likely hidden state sequence
- Compute the probability of a given observation sequence
- Given an observation sequence and set of possible models, which model most closely fits the data?

Most likely hidden state sequence

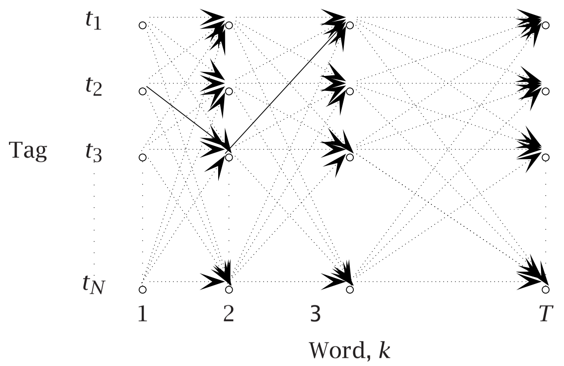
- Given $O = (o_1, \ldots, o_T)$ and model $\mu = (A, B, \Pi)$
- We want to find:

  $\arg\max_X P(X|O, \mu) = \arg\max_X \frac{P(X, O|\mu)}{P(O|\mu)} = \arg\max_X P(X, O|\mu)$
- $P(O|X, \mu) = b_{x_1 o_1} b_{x_2 o_2} \cdots b_{x_T o_T}$
- $P(X|\mu) = \pi_{x_1} a_{x_1 x_2} a_{x_2 x_3} \cdots a_{x_{T-1} x_T}$
- $P(O, X|\mu) = P(O|X, \mu) P(X|\mu)$
- $\arg\max_X P(O, X|\mu) = \arg\max_{x_1 \cdots x_T} \prod_{t=1}^{T} a_{x_{t-1} x_t} b_{x_t o_t}$
- Problem: Exponential in sequence length!

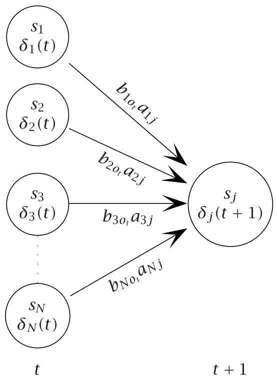
Dynamic Programming

- Efficient computation of this maximum: Viterbi algorithm
- Intuition: Probability of the first t observations is the same for all possible $t+1$ length state sequences.
- Define forward score
  $\delta_i(t) = \max_{x_1 \dots x_{t-1}} P(o_1 o_2 \cdots o_{t-1}, x_1 \cdots x_{t-1}, X_t = i|\mu)$
- $\delta_j(t+1) = \max_{i=1}^{N} \delta_i(t) b_{io_t} a_{ij}$
- Compute it recursively from beginning
- Remember best paths
- A version of Bayes Net most likely state inference

Trellis algorithms



Closeup of the computation at one node



$\delta_j(t+1) = \max_{i=1}^{N} \delta_i(t) b_{io_t} a_{ij}$

Viterbi algorithm (Viterbi 1967)

- Used to efficiently find the state sequence that gives the highest probability to the observed outputs
- A dynamic programming algorithm. Essentially the same except you do a max instead of a summation, and record the path taken:

$$\delta_{i+1}(t^j) = \max_{1 \le k \le T} [\delta_i(t^k) \times P(w_i|t^k) \times P(t^j|t^k)]$$

$$\psi_{i+1}(t^j) = \arg\max_{1 \le k \le T} [\delta_i(t^k) \times P(w_i|t^k) \times P(t^j|t^k)]$$

- This gives a best tag sequence for POS tagging
- (Note: this is different to finding the most likely tag for each time t!)