Natural Language Processing
Winter 2013

Hidden Markov Models

Luke Zettlemoyer - University of Washington

[Many slides from Dan Klein and Michael Collins]

Overview

- Hidden Markov Models
- Learning
  - Supervised: Maximum Likelihood
- Inference (or Decoding)
  - Viterbi
  - Forward Backward
- N-gram Taggers

Pairs of Sequences

- Consider the problem of jointly modeling a pair of strings
  - E.g.: part of speech tagging

DT NNP NN VBD VBN RP NN NNS
The Georgia branch had taken on loan commitments …

DT NN IN NN VBD NNS VBD
The average of interbank offered rates plummeted …

- Q: How do we map each word in the input sentence onto the appropriate label?
- A: We can learn a joint distribution:

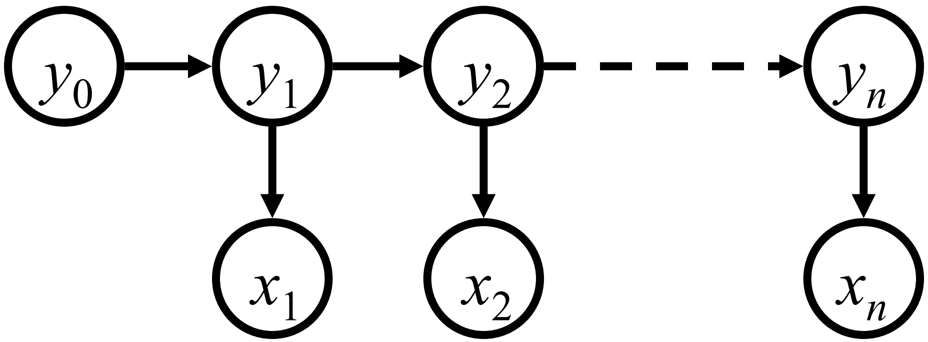
$$p(x_1 \dots x_n, y_1 \dots y_n)$$

  - And then compute the most likely assignment:

$$\arg\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

Classic Solution: HMMs

- We want a model of sequences y and observations x



$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

where y_0=*START* and we call q(y'|y) the transition distribution and e(x|y) the emission (or observation) distribution.

- Assumptions:
  - Tag/state sequence is generated by a markov model
  - Words are chosen independently, conditioned only on the tag/state
  - These are totally broken assumptions: why?

Example: POS Tagging

The Georgia branch had taken on loan commitments …

DT NNP NN VBD VBN RP NN NNS

- HMM Model:
  - States Y = {DT, NNP, NN, ... } are the POS tags
  - Observations X = V are words
  - Transition dist'n $q(y_i | y_{i-1})$ models the tag sequences
  - Emission dist'n $e(x_i | y_i)$ models words given their POS
- Q: How to we represent n-gram POS taggers?

Example: Chunking

- Goal: Segment text into spans with certain properties
- For example, named entities: PER, ORG, and LOC

Germany ’s representative to the European Union ’s veterinary committee Werner Zwingman said on Wednesday consumers should…

[Germany]$_{\mathrm{LOC}}$ ’s representative to the [European Union]$_{\mathrm{ORG}}$ ’s veterinary committee [Werner Zwingman]$_{\mathrm{PER}}$ said on Wednesday consumers should…

- Q: Is this a tagging problem?

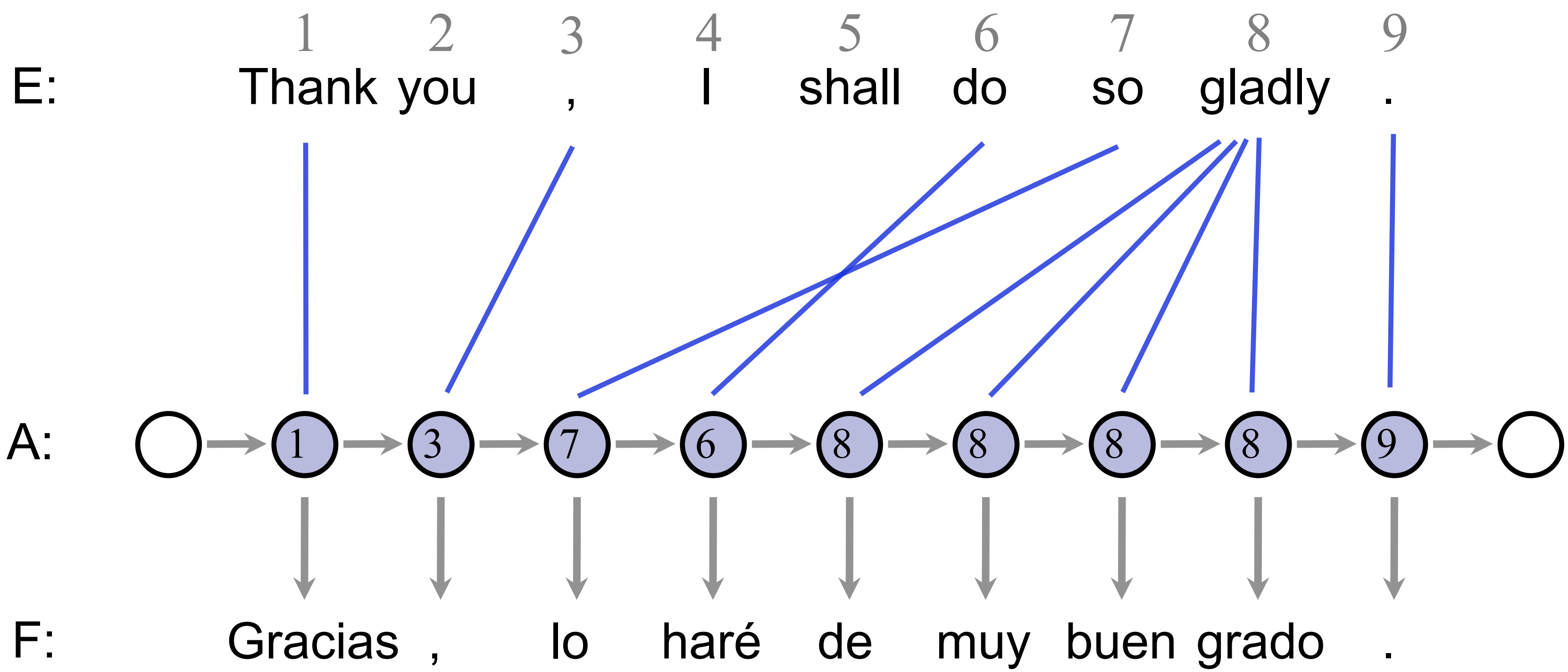
Example: Chunking

[Germany]$_{\text{LOC}}$ ’s representative to the [European Union]$_{\text{ORG}}$ ’s veterinary committee [Werner Zwingman]$_{\text{PER}}$ said on Wednesday consumers should…

Germany/BL ’s/NA representative/NA to/NA the/NA European/BO Union/CO ’s/NA veterinary/NA committee/NA Werner/BP Zwingman/CP said/NA on/NA Wednesday/NA consumers/NA should/NA…

- HMM Model:
  - States Y = {NA,BL,CL,BO,CO,BP,CP} represent beginnings (BL,BO,BP) and continuations (CL,CO,CP) of chunks, as well as other words (NA)
  - Observations X = V are words
  - Transition dist’n $q(y_i | y_{i-1})$ models the tag sequences
  - Emission dist’n $e(x_i | y_i)$ models words given their type

Example: HMM Translation Model



Model Parameters

Emissions: $e(F_1 = \text{Gracias} \mid E_{A_1} = \text{Thank})$ Transitions: $p(A_2 = 3 \mid A_1 = 1)$

HMM Inference and Learning

- Learning
  - Maximum likelihood: transitions q and emissions e

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Inference (linear time in sentence length!)
  - Viterbi:

$$y^* = \arg\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

  - Forward Backward:

$$p(x_1 \dots x_n, y_i) = \sum_{y_1 \dots y_{i-1}} \sum_{y_{i+1} \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

Learning: Maximum Likelihood

$$p(x_1 \dots x_n, y_1 \dots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Learning
  - Maximum likelihood methods for estimating transitions q and emissions e

$$q_{ML}(y_i|y_{i-1}) = \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} \qquad e_{ML}(x|y) = \frac{c(y, x)}{c(y)}$$

  - Will these estimates be high quality?
    - Which is likely to be more sparse, q or e?
  - Can use all of the same smoothing tricks we saw for language models!

Learning: Low Frequency Words

$$p(x_1 \ldots x_n, y_1 \ldots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Typically, linear interpolation works well for transitions

$$q(y_i|y_{i-1}) = \lambda_1 q_{ML}(y_i|y_{i-1}) + \lambda_2 q_{ML}(y_i)$$

- However, other approaches used for emissions
  - Step 1: Split the vocabulary
    - *Frequent words:* appear more than M (often 5) times
    - *Low frequency:* everything else
  - Step 2: Map each low frequency word to one of a small, finite set of possibilities
    - For example, based on prefixes, suffixes, etc.
  - Step 3: Learn model for this new space of possible word sequences

Low Frequency Words: An Example

Named Entity Recognition [Bickel et. al, 1999]

- Used the following word classes for infrequent words:

Word class	Example	Intuition
twoDigitNum	90	Two digit year
fourDigitNum	1990	Four digit year
containsDigitAndAlpha	A8956-67	Product code
containsDigitAndDash	09-96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
containsDigitAndPeriod	1.00	Monetary amount,percentage
othernum	456789	Other number
allCaps	BBN	Organization
capPeriod	M.	Person name initial
firstWord	first word of sentence	no useful capitalization information
initCap	Sally	Capitalized word
lowercase	can	Uncapitalized word
other	,	Punctuation marks, all other words

Low Frequency Words: An Example

- Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA quarter/NA results/NA ./NA

- *firstword*/NA soared/NA at/NA *initCap*/SC Co./CC ,/NA easily/NA *lowercase*/NA forecasts/NA on/NA *initCap*/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP *initCap*/CP announced/NA first/NA quarter/NA results/NA ./NA

NA = No entity
SC = Start Company
CC = Continue Company
SL = Start Location
CL = Continue Location
…

Inference (Decoding)

- Problem: find the most likely (Viterbi) sequence under the model

$$\arg\max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$$

- Given model parameters, we can score any sequence pair

NNP	VBZ	NN	NNS	CD	NN	.
Fed	raises	interest	rates	0.5	percent	.

q(NNP|♦) e(Fed|NNP) q(VBZ|NNP) e(raises|VBZ) q(NN|VBZ)…..

- In principle, we're done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

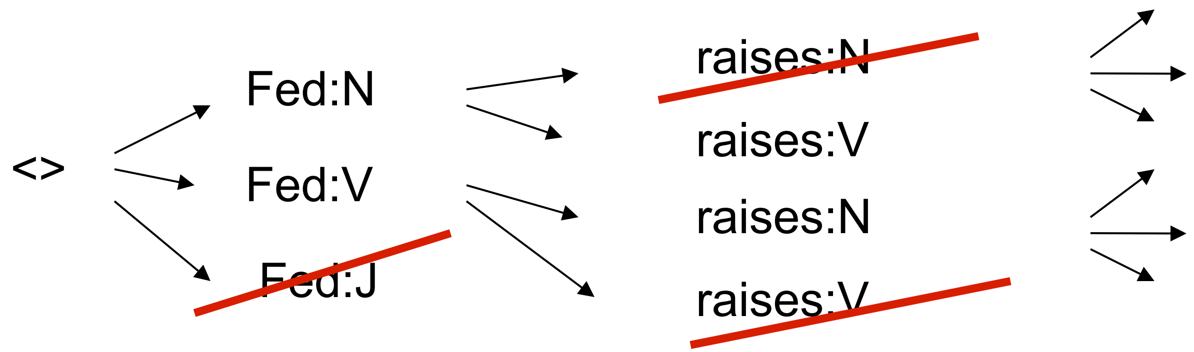
NNP VBZ NN NNS CD NN ⇨ logP = -23

NNP NNS NN NNS CD NN ⇨ logP = -29

NNP VBZ VB NNS CD NN ⇨ logP = -27

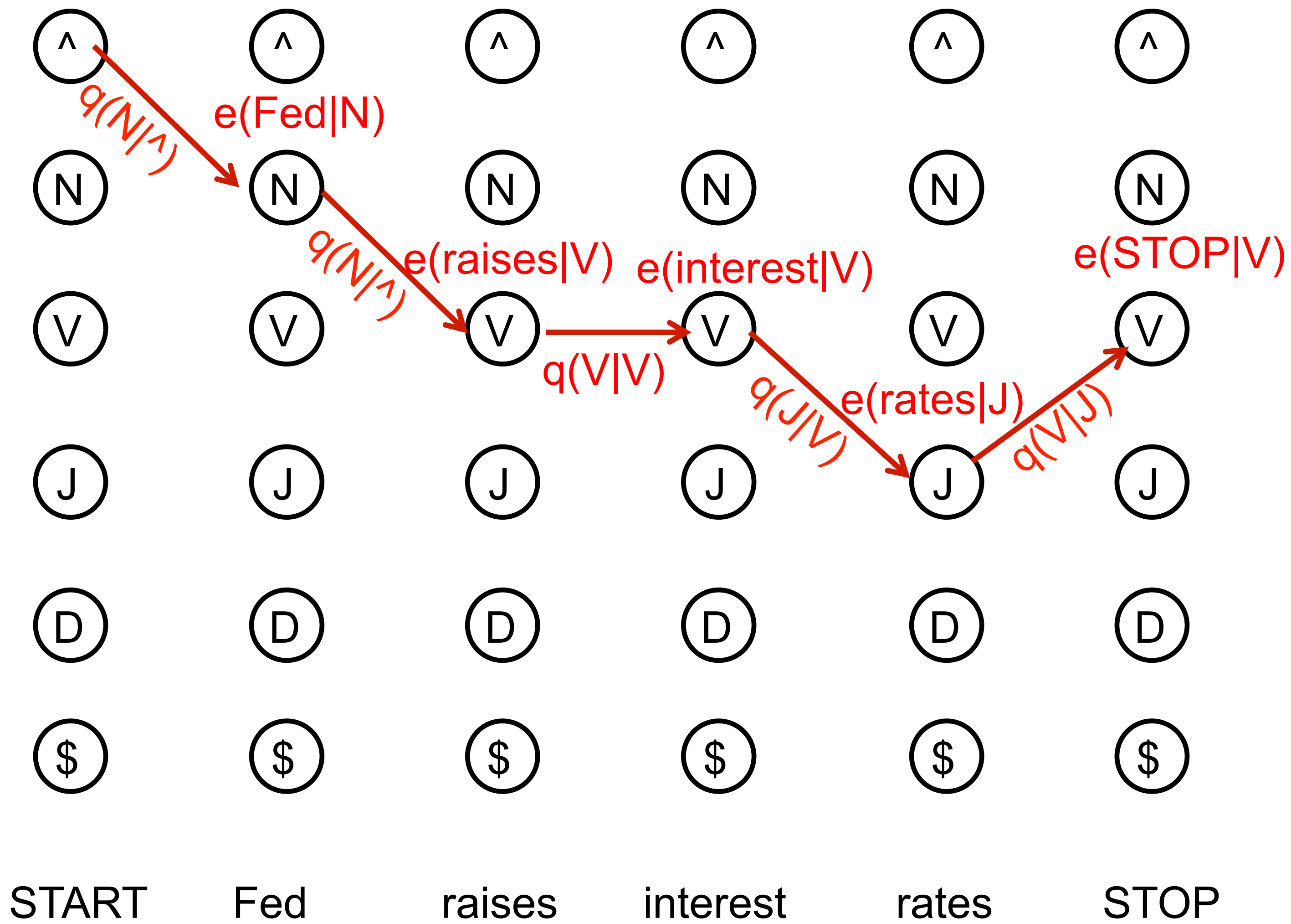
Finding the Best Trajectory

- Too many trajectories (state sequences) to list
- Option 1: Beam Search



  - A beam is a set of partial hypotheses
  - Start with just the single empty trajectory
  - At each derivation step:
    - Consider all continuations of previous hypotheses
    - Discard most, keep top k
- Beam search works ok in practice
  - … but sometimes you want the optimal answer
  - … and there's usually a better option than naïve beams

The State Lattice / Trellis



Dynamic Programming!

$$\arg\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

- Define $\pi(i, y_i)$ to be the max score of a sequence of length i ending in tag y_i

$$\pi(i, y_i) = \max_{y_1 \cdots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

$$= \max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \max_{y_1 \cdots y_{i-2}} p(x_1 \dots x_{i-1}, y_1 \dots y_{i-1})$$

$$= \max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1})$$

- We now have an efficient algorithm. Start with i=0 and work your way to the end of the sentence!

The Viterbi Algorithm

- Dynamic program for computing (for all i)

$$\pi(i, y_i) = \max_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

  - Iterative computation

$$\pi(0, y_0) = \begin{cases} 1 \text{ if } y_0 == START \\ 0 \text{ otherwise} \end{cases}$$

For i = 1 ... n:

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1})$$

  - Also, store back pointers

$$bp(i, y_i) = \arg\max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1})$$

The Viterbi Algorithm: Runtime

- Linear in sentence length n
- Polynomial in the number of possible tags |K|

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

- Specifically:

$$O(n|\mathcal{K}|) \text{ entries in } \pi(i, y_i)$$

$$O(|\mathcal{K}|) \text{ time to compute each } \pi(i, y_i)$$

- Total runtime: $O(n|\mathcal{K}|^2)$

- Q: Is this a practical algorithm?
- A: depends on |K|….

Marginal Inference

- Problem: find the marginal probability of each tag for y_i

$$p(x_1 \dots x_n, y_i) = \sum_{y_1 \dots y_{i-1}} \sum_{y_{i+1} \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$$

- Given model parameters, we can score any sequence pair

NNP	VBZ	NN	NNS	CD	NN	.
Fed	raises	interest	rates	0.5	percent	.

q(NNP|♦) e(Fed|NNP) q(VBZ|NNP) e(raises|VBZ) q(NN|VBZ)…..

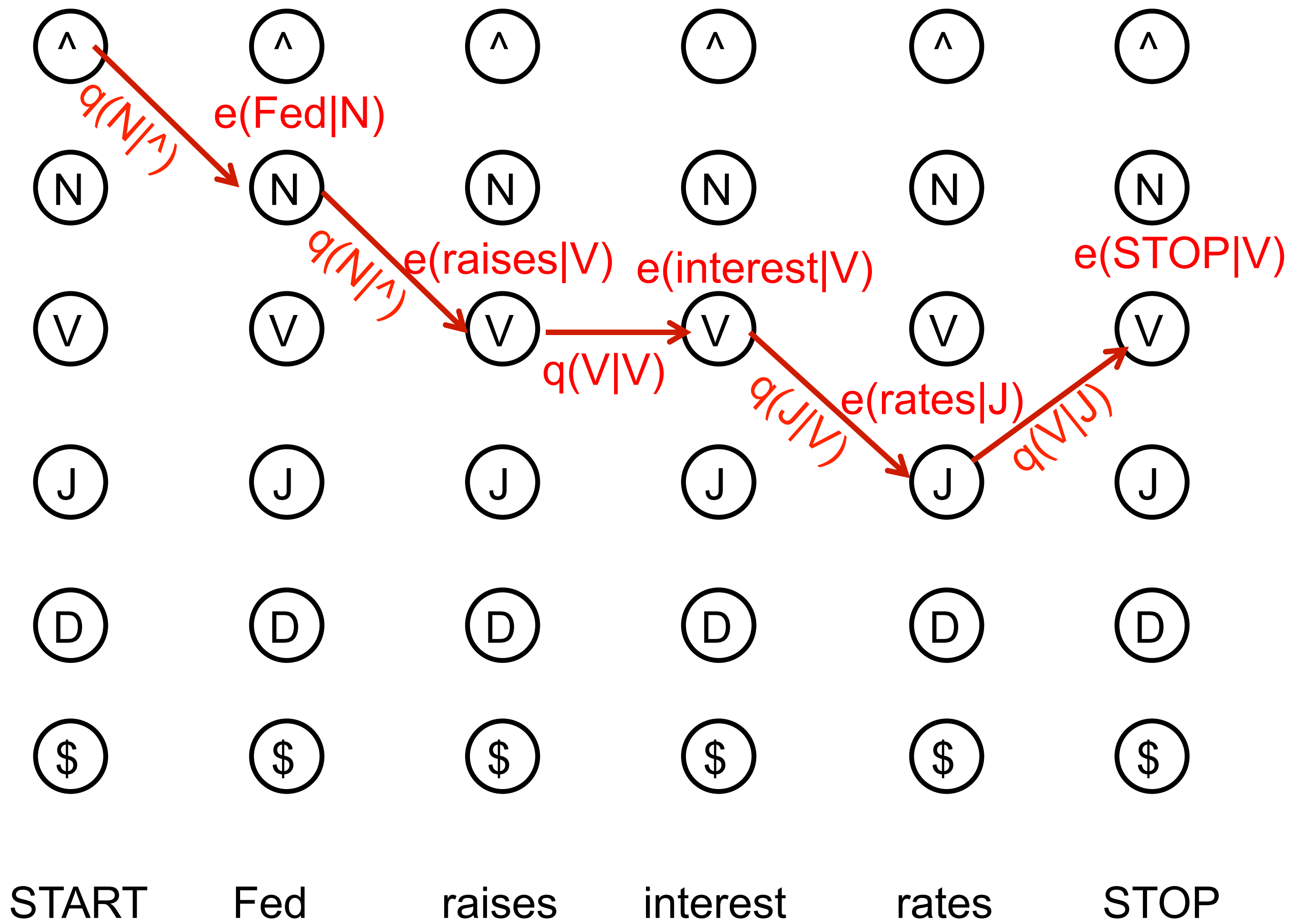
- In principle, we're done – list all possible tag sequences, score each one, sum over all of the possible values for y_i

NNP VBZ NN NNS CD NN ⇨ logP = -23

NNP NNS NN NNS CD NN ⇨ logP = -29

NNP VBZ VB NNS CD NN ⇨ logP = -27

The State Lattice / Trellis



Dynamic Programming!

$$p(x_1 \dots x_n, y_i) = p(x_i \dots x_i, y_i) p(x_{i+1} \dots x_n | y_i)$$

- Sum over all paths, on both sides of each y_i

$$\alpha(i, y_i) = p(x_1 \dots x_i, y_i) = \sum_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

$$= \sum_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \alpha(i-1, y_{i-1})$$

$$\beta(i, y_i) = p(x_{i+1} \dots x_n | y_i) = \sum_{y_{i+1} \dots y_n} p(x_{i+1} \dots x_n, y_{i+1} \dots y_n)$$

$$= \sum_{y_{i+1}} e(x_{i+1} | y_{i+1}) q(y_{i+1} | y_i) \beta(i+1, y_{i+1})$$

Forward Backward Algorithm

- Two passes: one forward, one back
  - Forward:

$$\alpha(0, y_0) = \begin{cases} 1 \text{ if } y_0 == START \\ 0 \text{ otherwise} \end{cases}$$

    - For i = 1 … n

$$\alpha(i, y_i) = \sum_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\alpha(i-1, y_{i-1})$$

  - Backward:

$$\beta(n, y_n) = \begin{cases} 1 \text{ if } y_n == STOP \\ 0 \text{ otherwise} \end{cases}$$

    - For i = n-1 ... 1

$$\beta(i, y_i) = \sum_{y_{i+1}} e(x_{i+1}|y_{i+1})q(y_{i+1}|y_i)\beta(i+1, y_{i+1})$$

Forward Backward: Runtime

- Linear in sentence length n
- Polynomial in the number of possible tags |K|

$$\alpha(i, y_i) = \sum_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\alpha(i-1, y_{i-1})$$

$$\beta(i, y_i) = \sum_{y_{i+1}} e(x_{i+1}|y_{i+1})q(y_{i+1}|y_i)\beta(i+1, y_{i+1})$$

- Specifically:

$$O(n|\mathcal{K}|) \text{ entries in } \alpha(i, y_i) \text{ and } \beta(i, y_i)$$

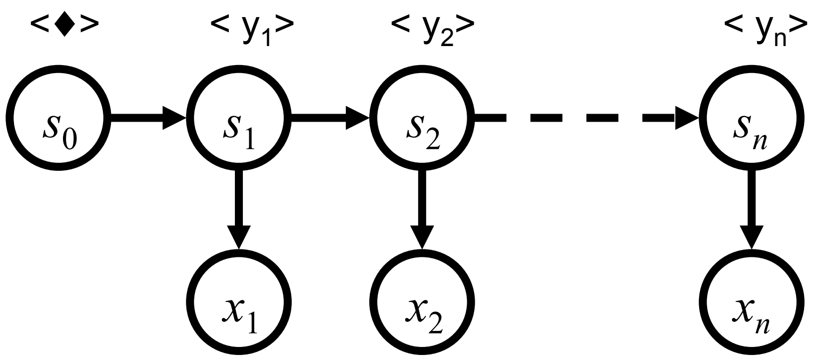
$$O(|\mathcal{K}|) \text{ time to compute each entry}$$

- Total runtime: $O(n|\mathcal{K}|^2)$

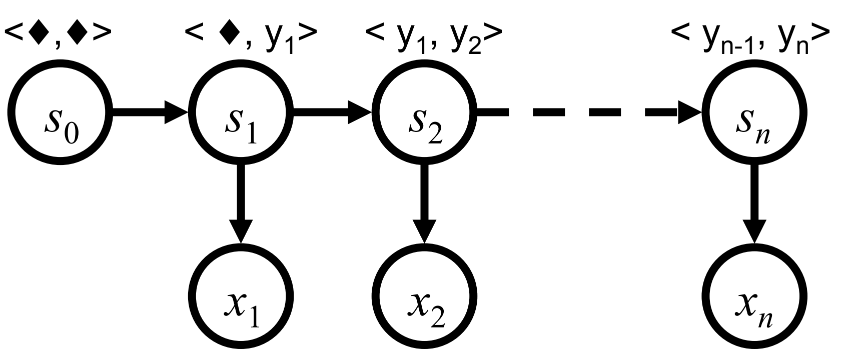
- Q: How does this compare to Viterbi?
- A: Exactly the same!!! (actually x2, a constant factor…)

What about n-gram Taggers?

- States encode what is relevant about the past
- Transitions P(s|s') encode well-formed tag sequences
  - In a bigram tagger, states = tags



  - In a trigram tagger, states = tag pairs



The State Lattice / Trellis