CSE P 517
Natural Language Processing
Winter 2021

Hidden Markov Models

Yejin Choi

University of Washington

[Many slides from Dan Klein, Michael Collins, Luke Zettlemoyer]

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:

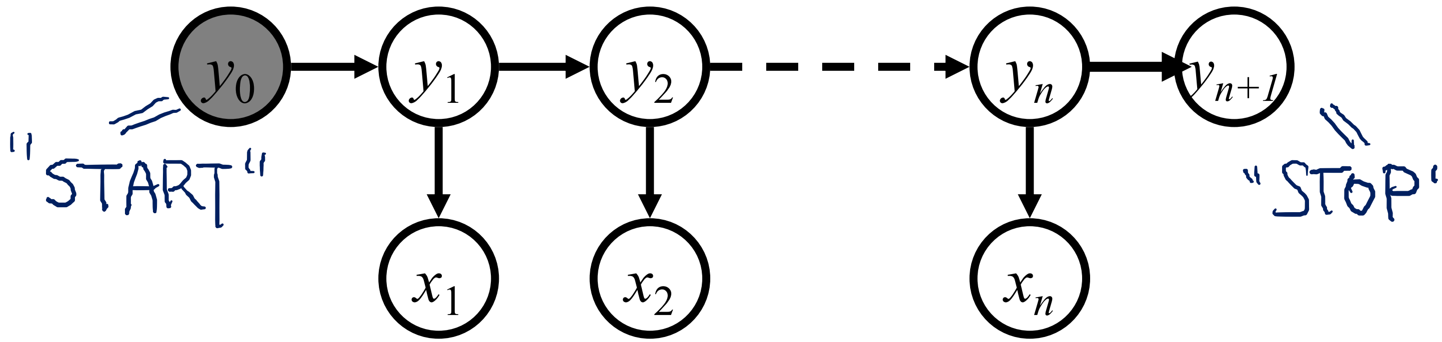
$$p(x_1 \dots x_n, y_1 \dots y_n)$$

  - 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



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

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

- Q: Is this a tagging problem?

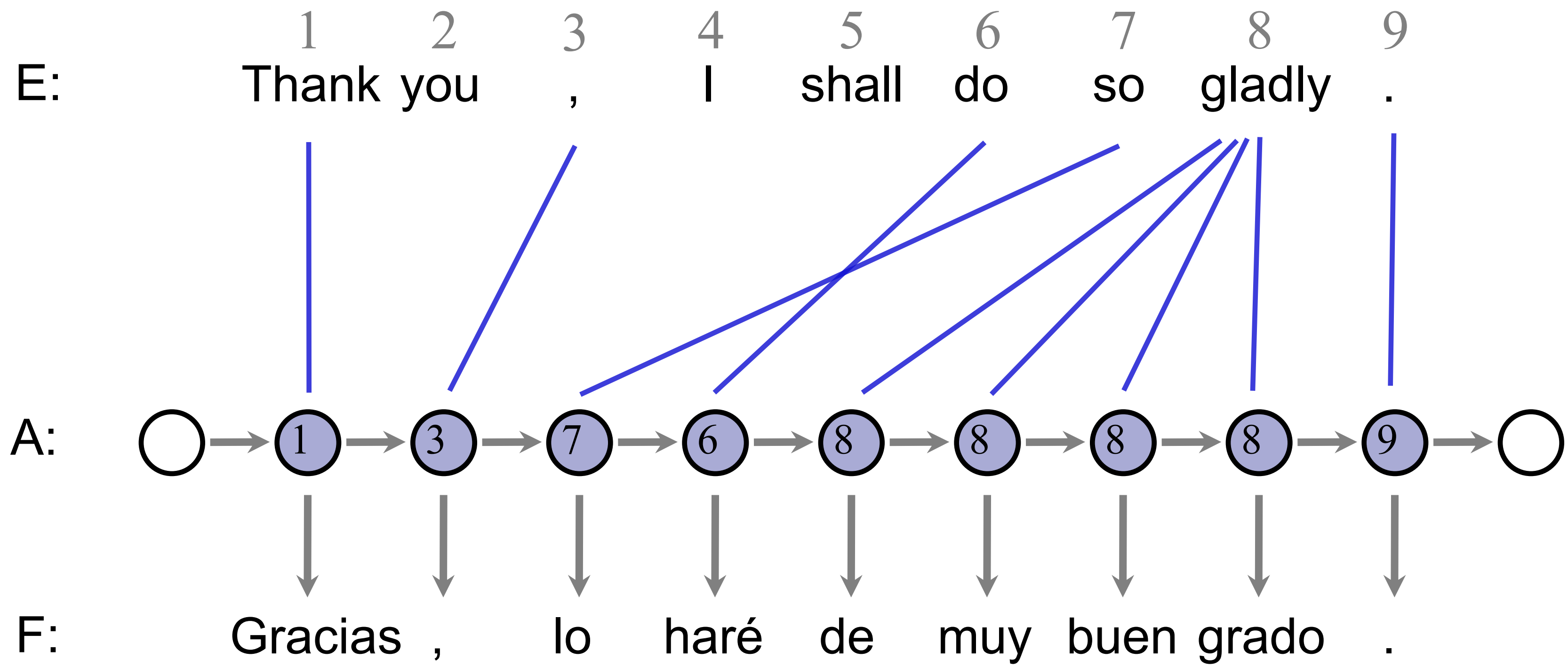
Example: Chunking

[Germany]$_{LOC}$'s representative to the [European Union]$_{ORG}$'s veterinary committee [Werner Zwingman]$_{PER}$ 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

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

- Inference (linear time in sentence length!)
  - Viterbi: $y* = \underset{y_1 \ldots y_n}{\operatorname{argmax}}\ p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$

    $\text{where } y_{n+1} = \text{STOP}$

  - Forward Backward:

$$p(x_1 \ldots x_n, y_i) =$$

Why on the earth forward-backward

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

?

Why learn forward-backward

1. It's a subroutine inside EM for unsupervised learning of sequence labeling (HMMs)
   - To replace actual counts with expected counts

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

2. It generalizes to inside-outside algorithm for unsupervised learning of trees (PCFGs)
3. It's also a subroutine when training linear-chain Conditional Random Fields

[Inside-outside and forward-backward algorithms are just backprop](#).
Jason Eisner (2016).
In *EMNLP Workshop on Structured Prediction for NLP*.

Inside-Outside & Forward-Backward Algorithms are just Backprop

(tutorial paper)

Jason Eisner

Johns Hopkins University

Center for Language and Speech Processing

"The inside-outside algorithm is the hardest algorithm I know."

– a senior NLP researcher, in the 1990's

β α

STRUCTURED ATTENTION NETWORKS

Yoon Kim* **Carl Denton*** **Luong Hoang** **Alexander M. Rush**
{yoonkim@seas,carldenton@college,lhoang@g,srush@seas}.harvard.edu
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, USA

procedure FORWARDBACKWARD(θ)

$\alpha[0, \langle t \rangle] \leftarrow 0$

$\beta[n+1, \langle t \rangle] \leftarrow 0$

for $i = 1, \dots, n; c \in \mathcal{C}$ **do**

$\alpha[i, c] \leftarrow \bigoplus_y \alpha[i-1, y] \otimes \theta_{i-1,i}(y, c)$

for $i = n, \dots, 1; c \in \mathcal{C}$ **do**

$\beta[i, c] \leftarrow \bigoplus_y \beta[i+1, y] \otimes \theta_{i,i+1}(c, y)$

$A \leftarrow \alpha[n+1, \langle t \rangle]$

procedure BACKPROPFORWARDBACKWARD$(\theta, p, \nabla_p^{\mathcal{L}})$

$\nabla_\alpha^{\mathcal{L}} \leftarrow \log p \otimes \log \nabla_p^{\mathcal{L}} \otimes \beta \otimes -A$

$\nabla_\beta^{\mathcal{L}} \leftarrow \log p \otimes \log \nabla_p^{\mathcal{L}} \otimes \alpha \otimes -A$

$\hat{\alpha}[0, \langle t \rangle] \leftarrow 0$

$\hat{\beta}[n+1, \langle t \rangle] \leftarrow 0$

for $i = n, \dots 1; c \in \mathcal{C}$ **do**

$\hat{\beta}[i, c] \leftarrow \nabla_\alpha^{\mathcal{L}}[i, c] \oplus \bigoplus_y \theta_{i,i+1}(c, y) \otimes \hat{\beta}[i+1, y]$

for $i = 1, \dots, n; c \in \mathcal{C}$ **do**

Google DeepMind Is Now Analysing *Magic* And *Hearthstone* Cards

Logan Booker
3/28/16 8:30pm • Filed to: GOOGLE

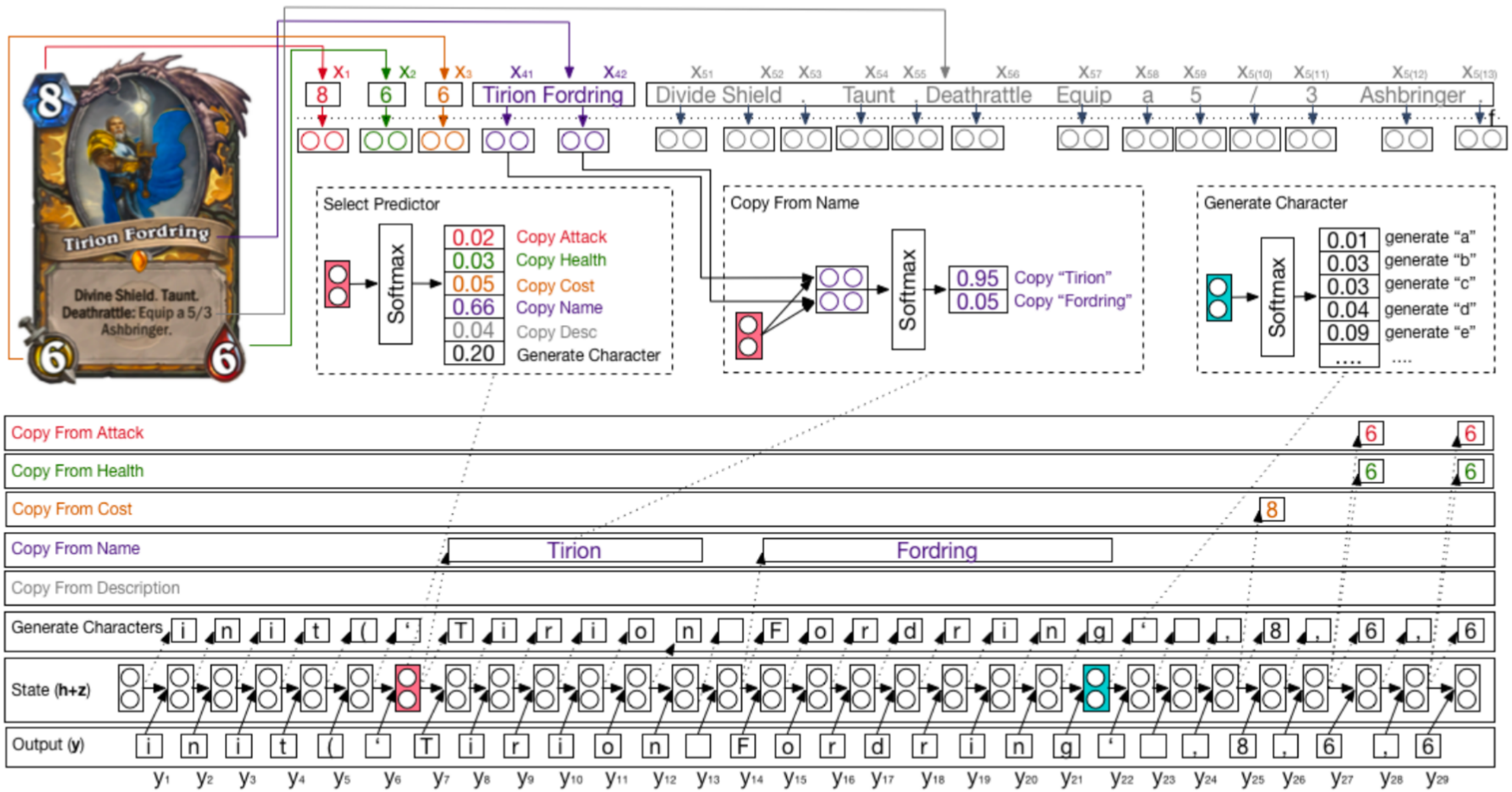


Figure 4: Generation process for the code `init('Tirion Fordring',8,6,6)` using LPNs.

Latent Predictor Networks for Code Generation

Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Andrew Senior, Fumin Wang, Phil Blunsom

ACL 2016

While the number of possible paths grows exponentially, α and β can be computed efficiently using the forward-backward algorithm for Semi-Markov models (Sarawagi and Cohen, 2005), where we associate $P(r_t \mid y_1..y_{t-1}, x)$ to edges and $P(s_t \mid y_1..y_{t-1}, x, r_t)$ to nodes in the Markov chain.

The derivative $\frac{\partial \log P(y|x)}{\partial P(s_t|y_1..y_{t-1},x,r_t)}$ can be computed using the same logic:

$$\frac{\partial \alpha_{t,s_t} P(s_t \mid y_1..y_{t-1}, x, r_t) \beta_{t+|s_t|-1} + \xi_{r_t}}{P(y \mid x) \partial P(s_t \mid y_1..y_{t-1}, x, r_t)} = \frac{\alpha_{t,r_t} \beta_{t+|s_t|-1}}{\alpha_{|y|+1}}$$

Learning: Maximum Likelihood

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

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

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

  - 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

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

- 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

$$y* = \underset{y_1 \ldots y_n}{\operatorname{argmax}}\ p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

- 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

Dynamic Programming!

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

$$y* = \underset{y_1 \ldots y_n}{\operatorname{argmax}} \; p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

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

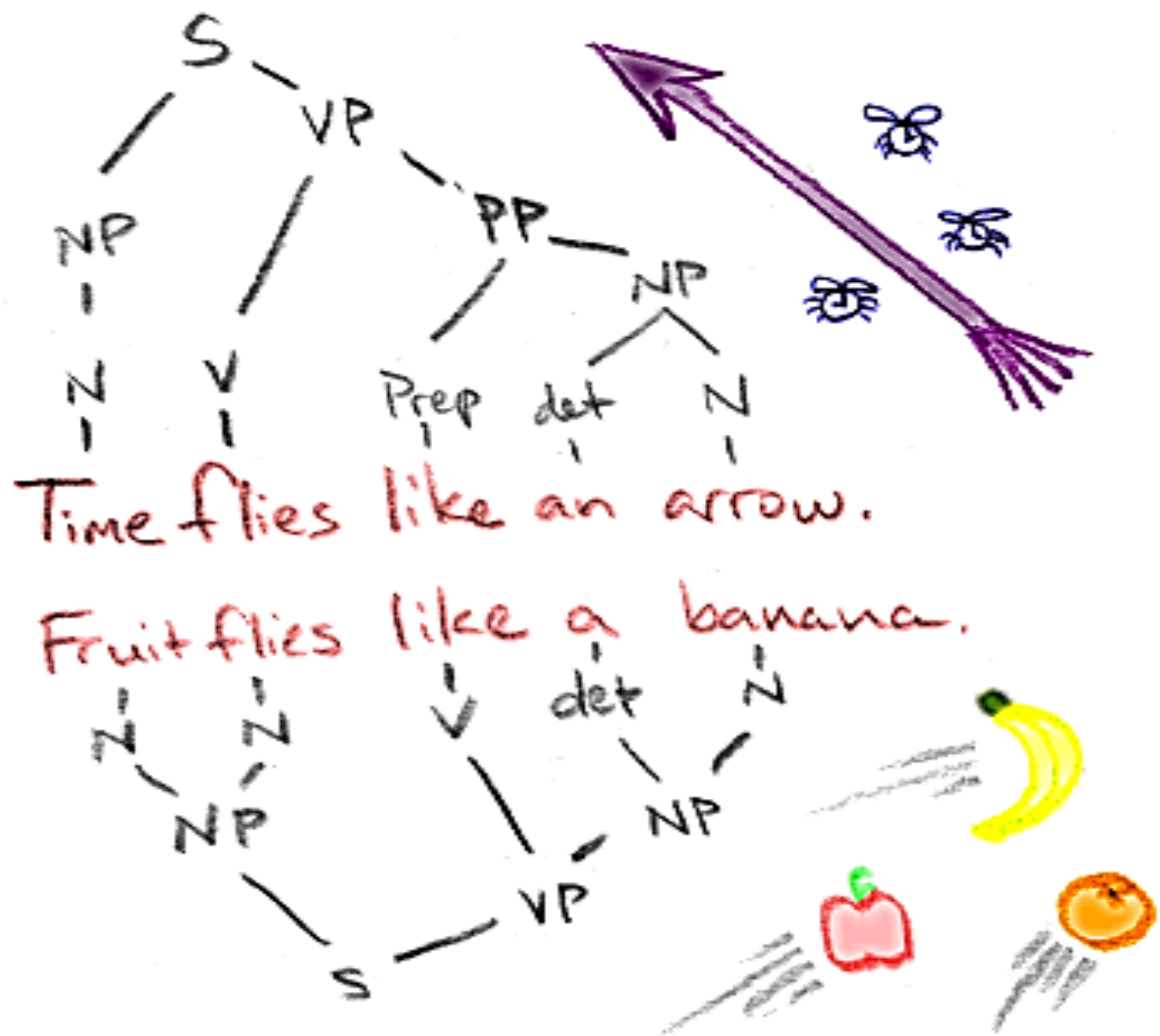
$$\pi(i, y_i) = \max_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

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

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

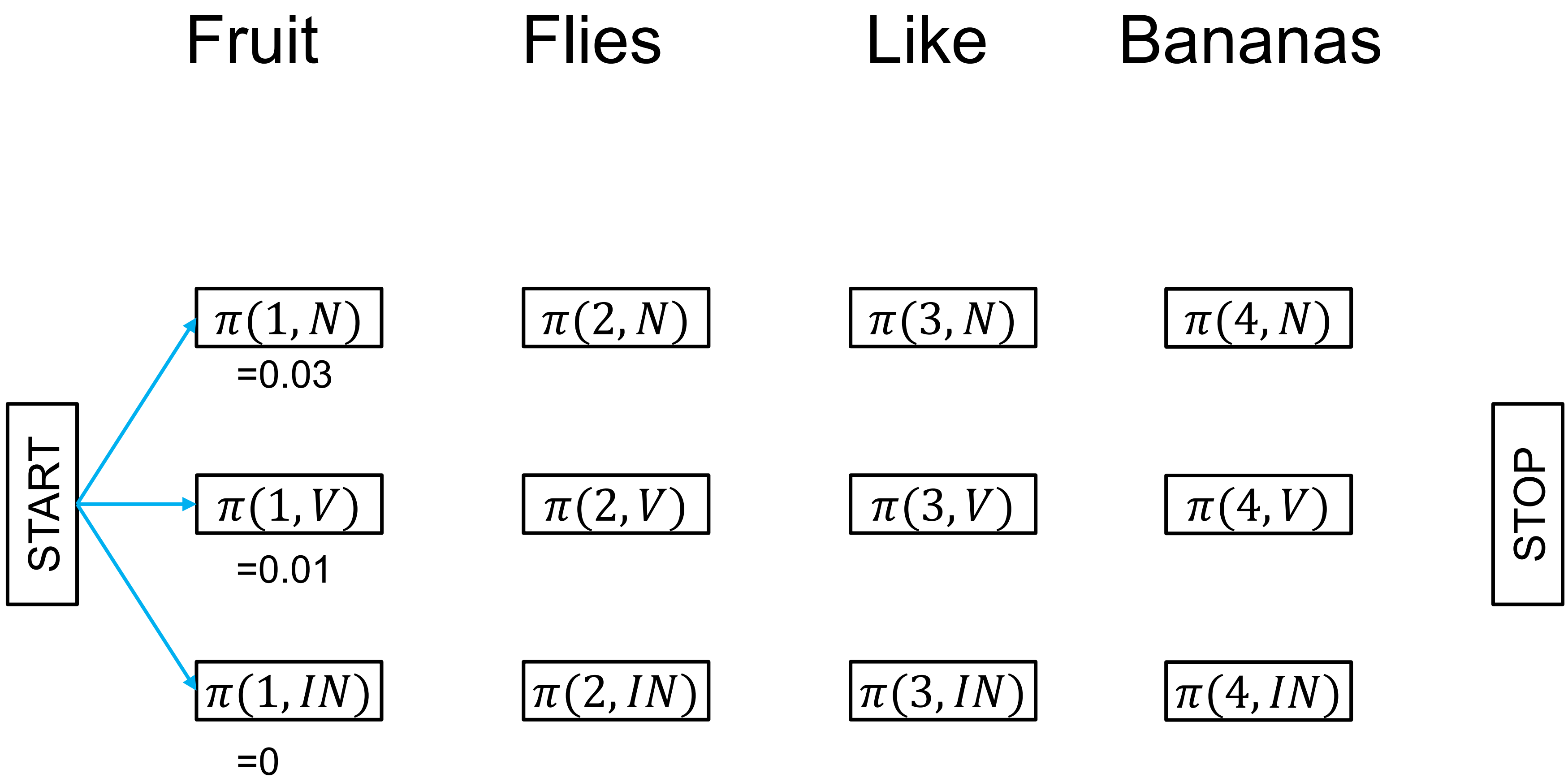
- We now have an efficient algorithm. Start with i=0 and work your way to the end of the sentence!

Time flies like an arrow; Fruit flies like a banana

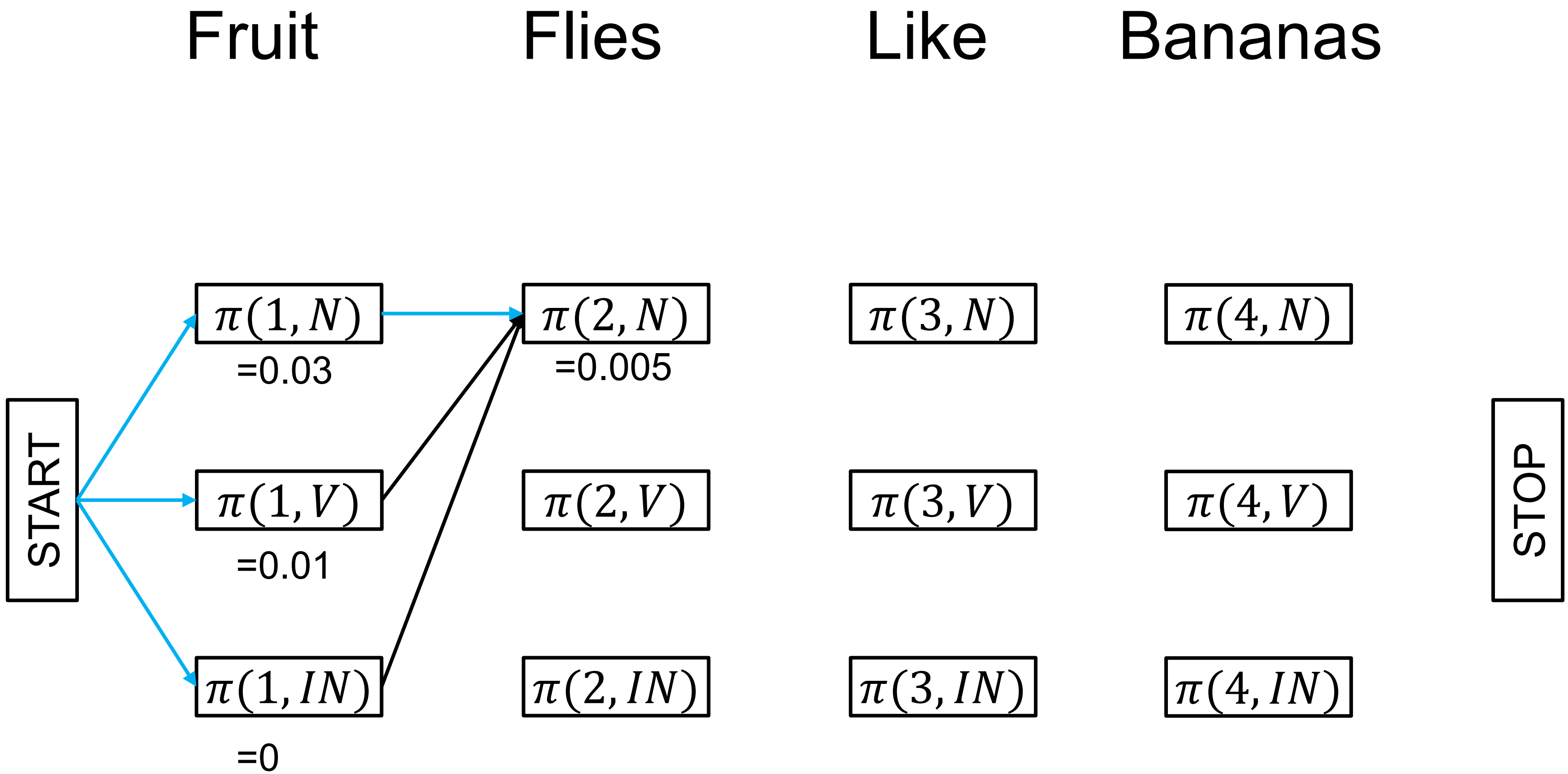


	Fruit	Flies	Like	Bananas	
	$\pi(1, N)$	$\pi(2, N)$	$\pi(3, N)$	$\pi(4, N)$	
START	$\pi(1, V)$	$\pi(2, V)$	$\pi(3, V)$	$\pi(4, V)$	STOP
	$\pi(1, IN)$	$\pi(2, IN)$	$\pi(3, IN)$	$\pi(4, IN)$	

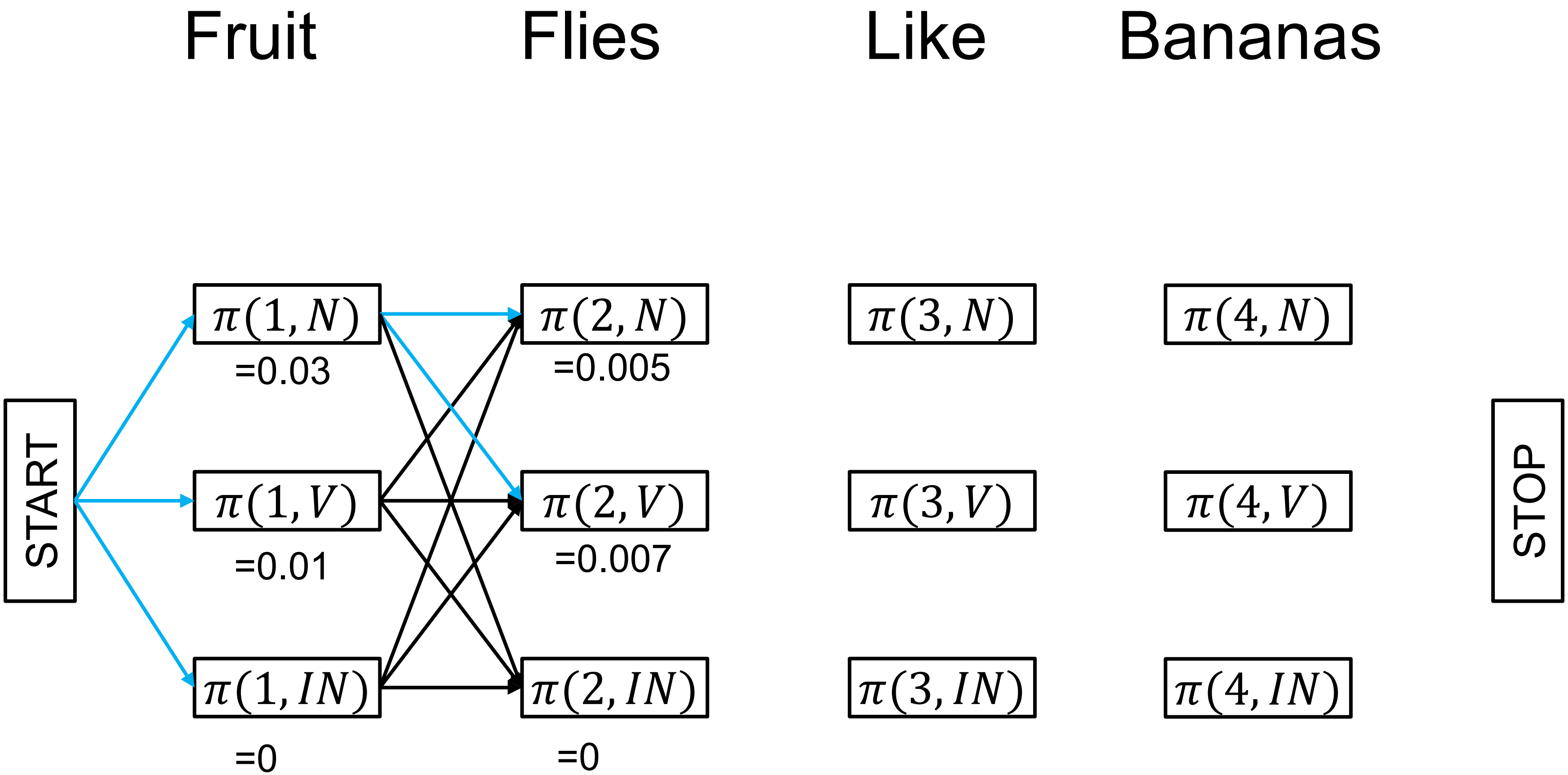
$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$



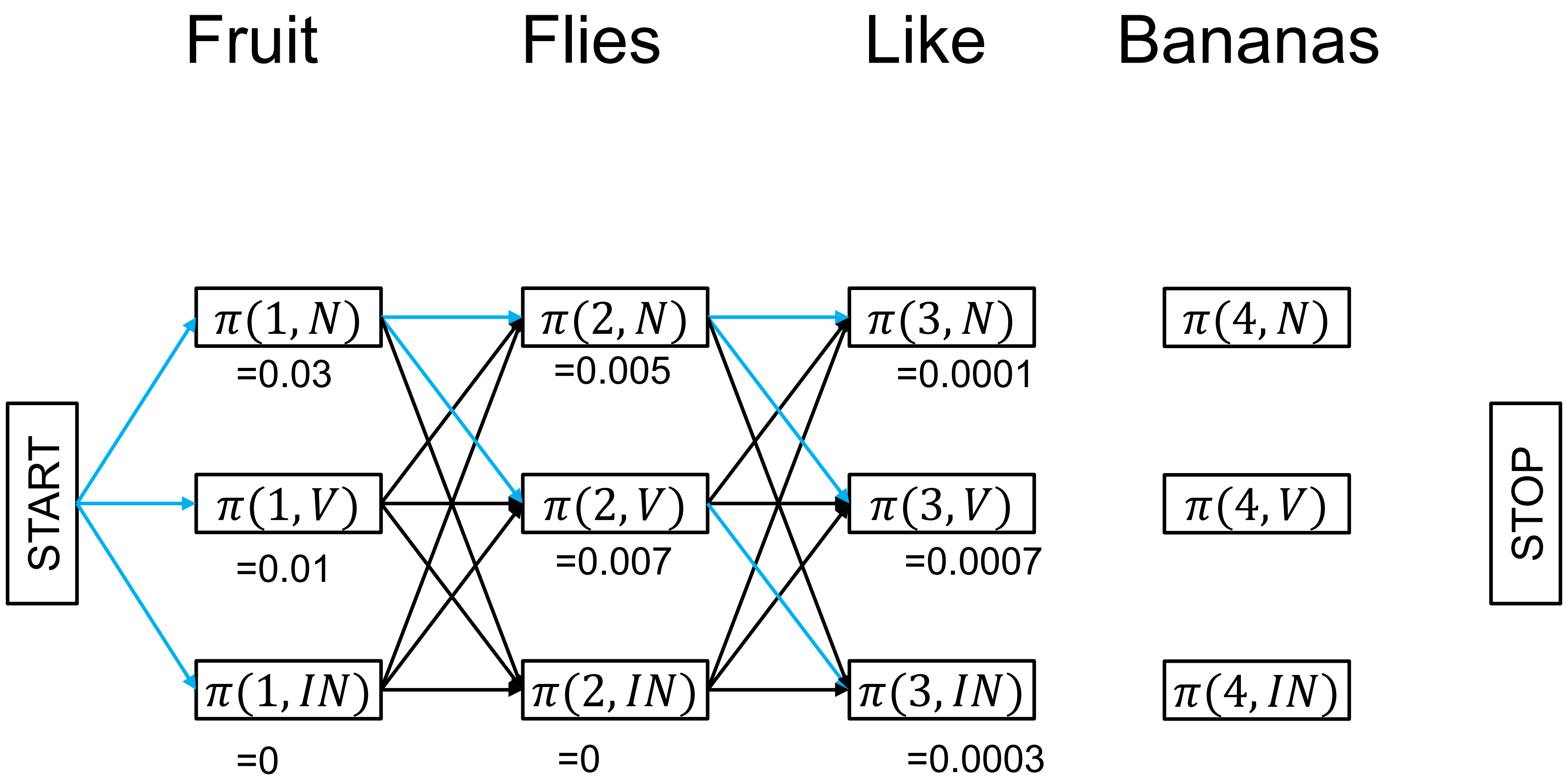
$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$



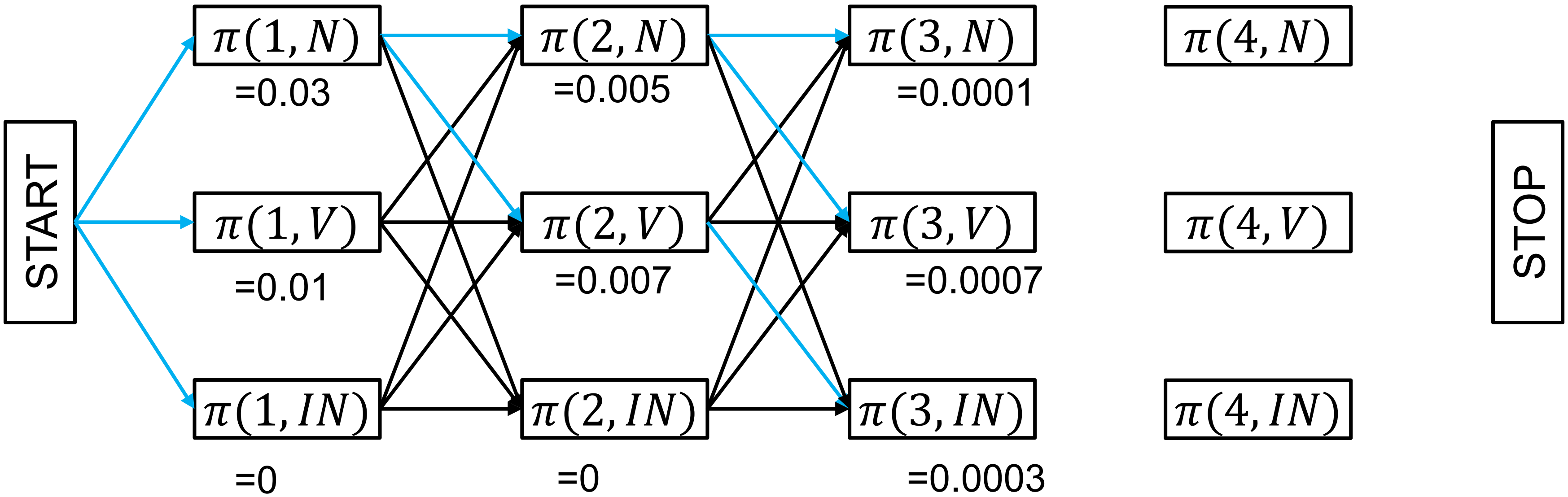
$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$



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



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

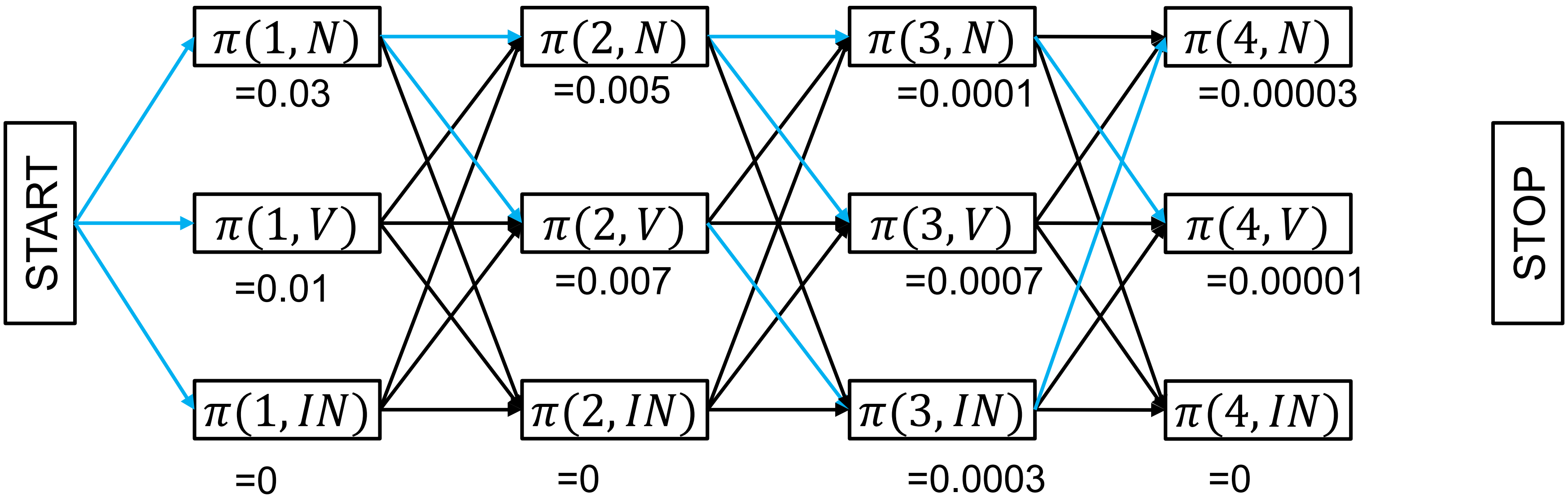


$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

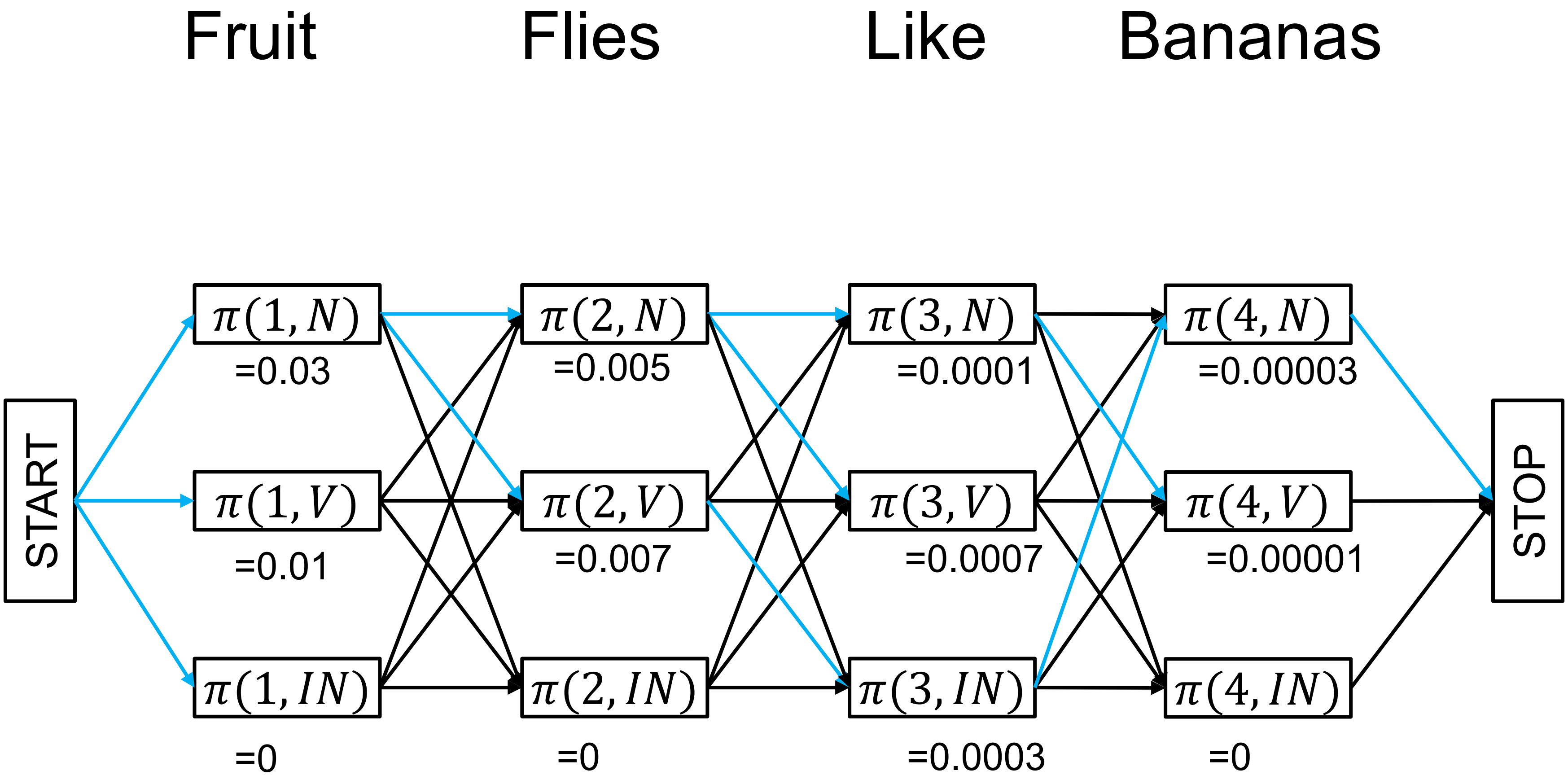
$$= \max_{y_1 \dots y_{i-2}} p(x_1 \dots x_{i-1}, y_1 \dots y_{i-1})$$

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

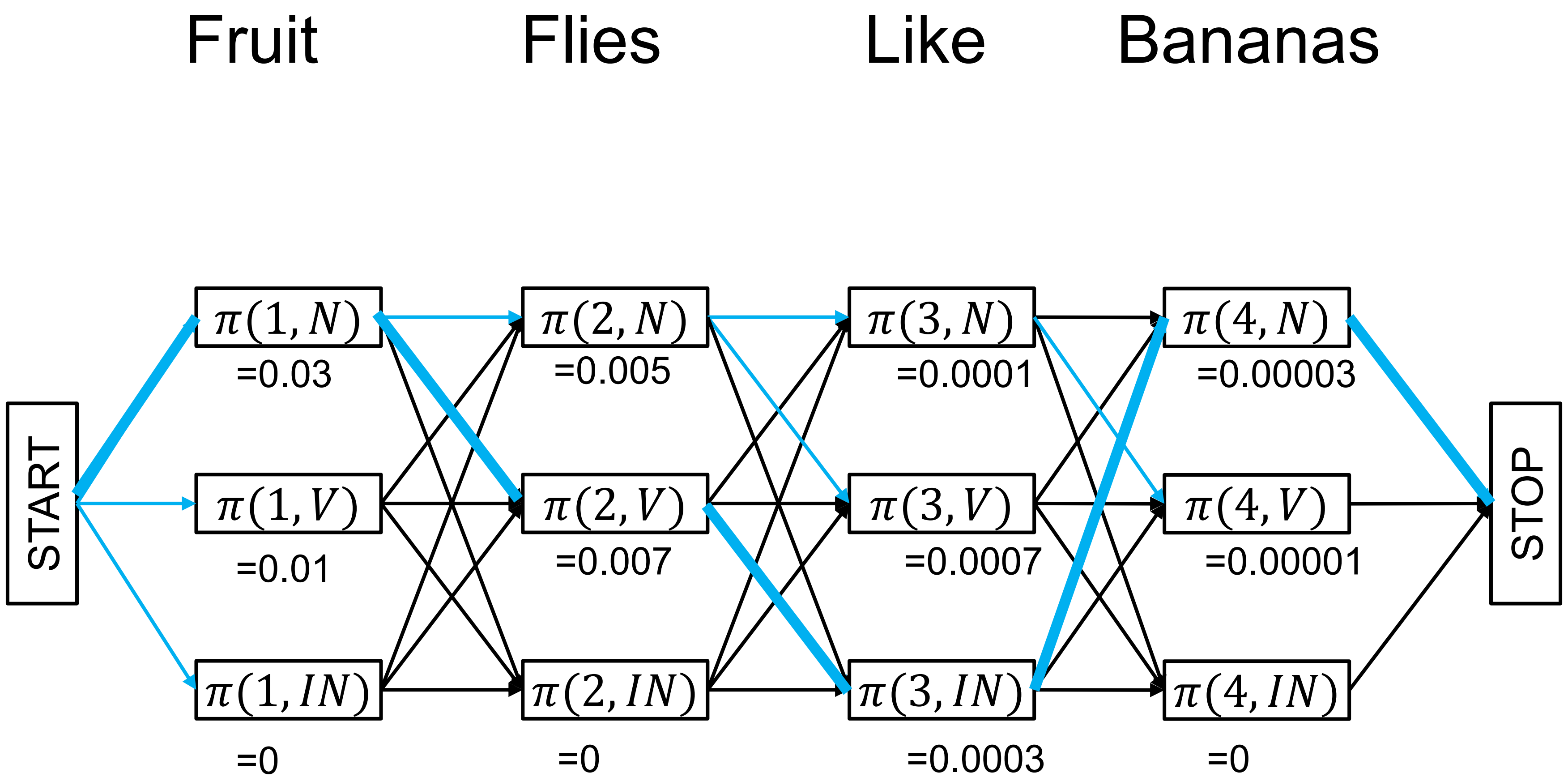
Fruit Flies Like Bananas



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

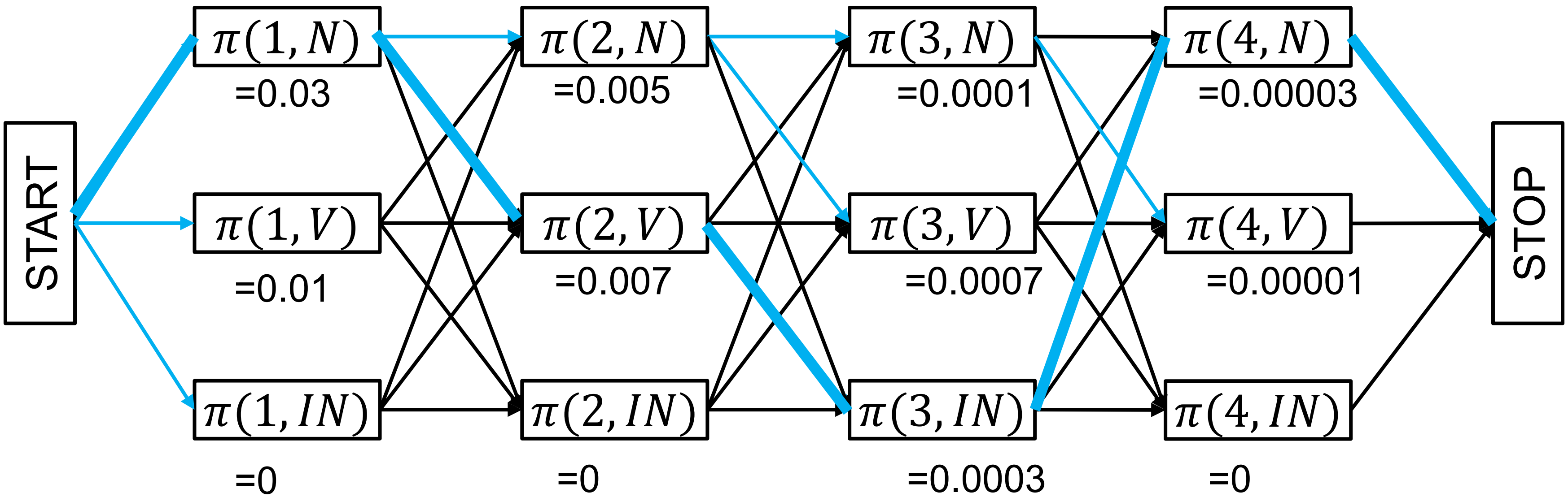


$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$



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

Why is this not a greedy algorithm?
Why does this find the max p(.)?
What is the runtime?



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

Dynamic Programming!

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

$$y* = \underset{y_1 \cdots y_n}{\operatorname{argmax}}\; p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

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

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

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

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

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

Viterbi Algorithm

Viterbi!

- Dynamic program for computing (for all i)

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

  - Iterative computation

$$\pi(0, y_0) =$$

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

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

- What is the final solution to $y* = \underset{y_1 \dots y_n}{\text{argmax}}\ p(x_1 \dots x_n, y_1 \dots y_{n+1})$?

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|….

Broader Context

- **Beam Search**: Viterbi decoding with K best sub-solutions (beam size = K)
- Viterbi algorithm - a special case of **max-product** algorithm
- Forward-backward - a special case of **sum-product** algorithm (**belief propagation** algorithm)
- Viterbi decoding can be also used with general graphical models (factor graphs, Markov Random Fields, Conditional Random Fields, …) with non-probabilistic scoring functions (potential functions).

Reflection

- Viterbi: why argmax over joint distribution?

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

- Why not this:

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

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

- Same thing!

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

Marginal Inference

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

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

- 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

Marginal Inference

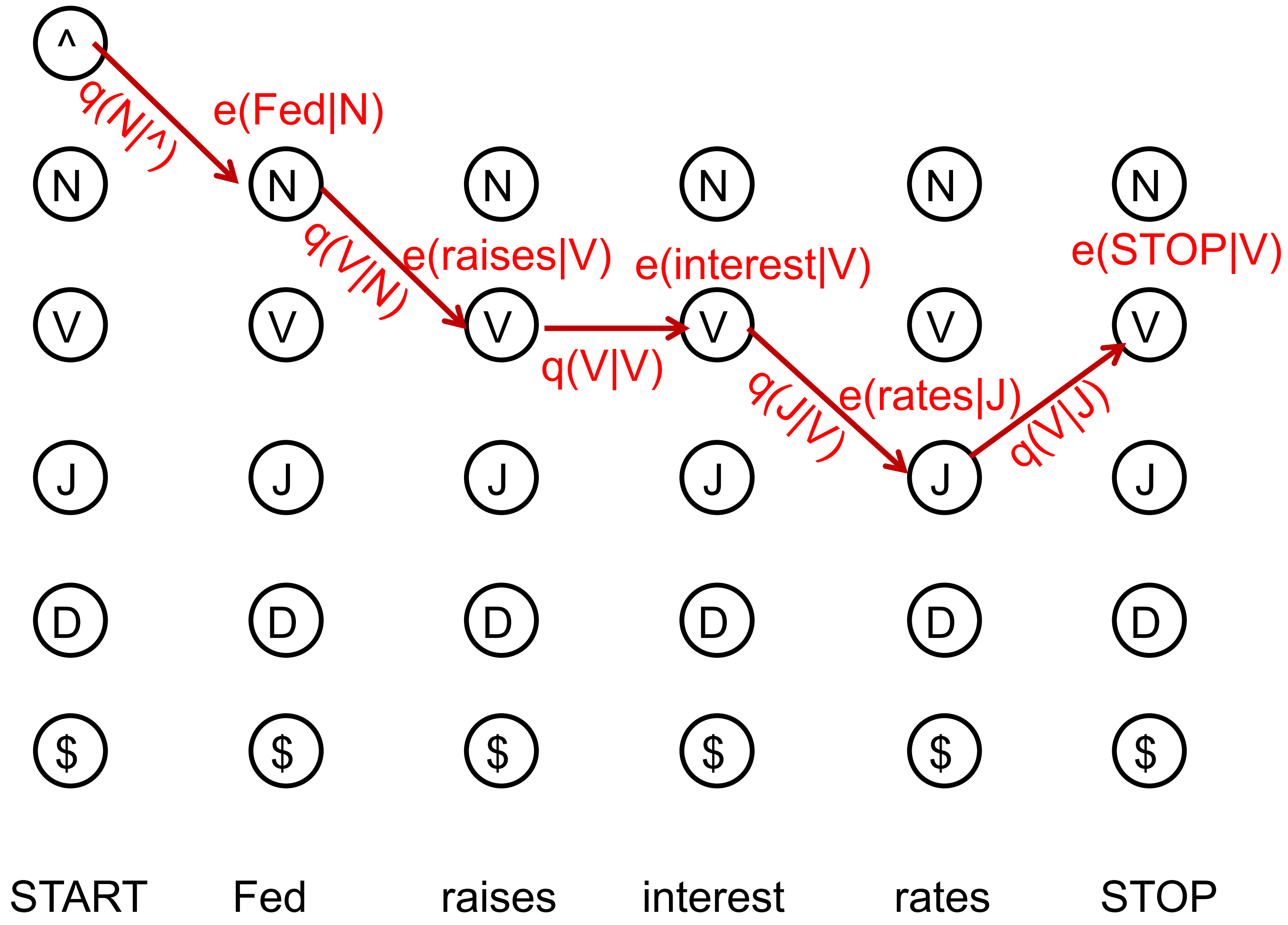
- Problem: find the marginal probability of each tag for y_i

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

Compare it to “Viterbi Inference”

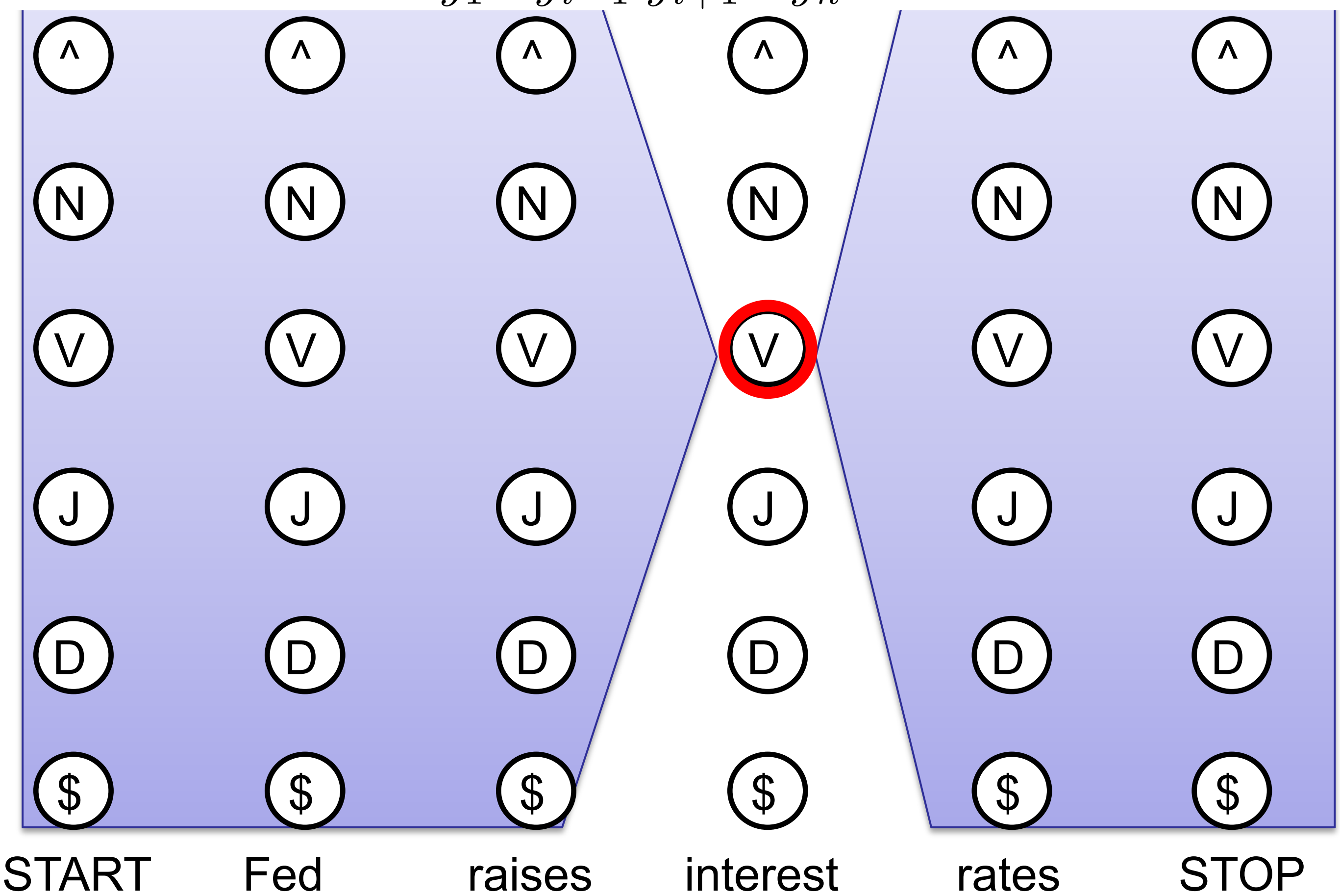
$$\pi(i, y_i) = \max_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

The State Lattice / Trellis: Viterbi



The State Lattice / Trellis: Marginal

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



Dynamic Programming!

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

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

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

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

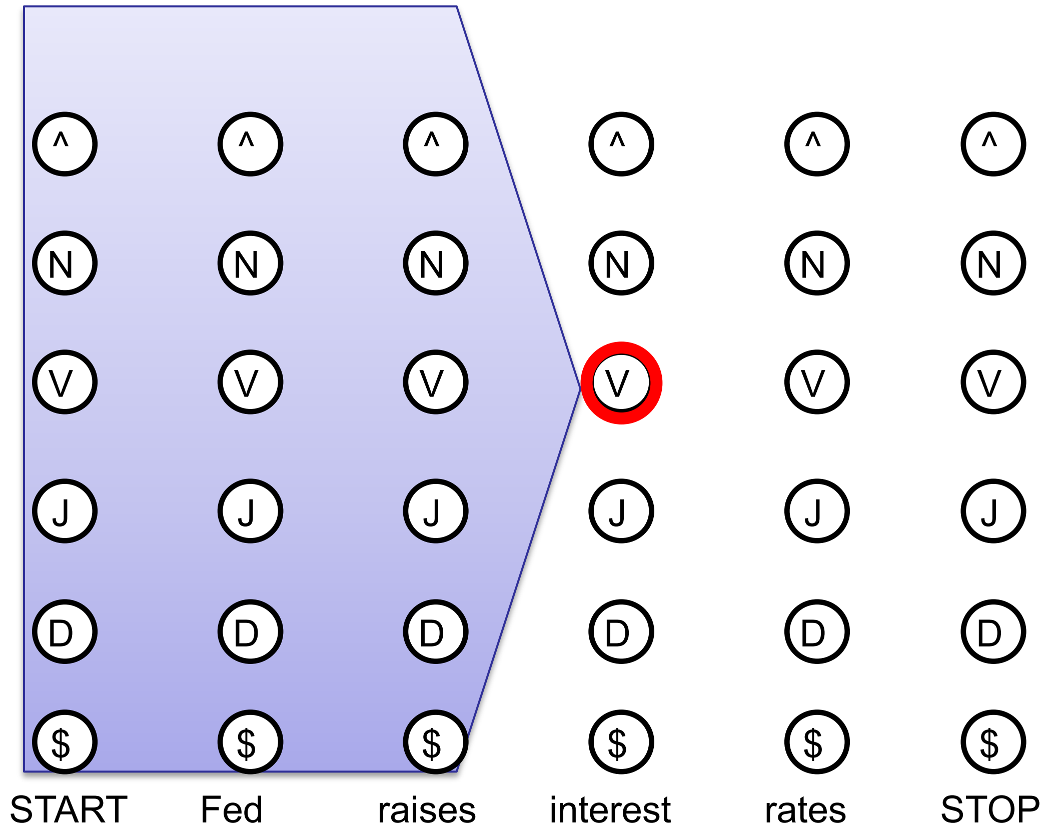
$$\beta(i, y_i) = p(x_{i+1} \dots x_n | y_i) = \sum_{y_{i+1} \dots y_n} p(x_{i+1} \dots x_n, y_{i+1} \dots y_{n+1} | y_i)$$

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

The State Lattice / Trellis: Forward

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

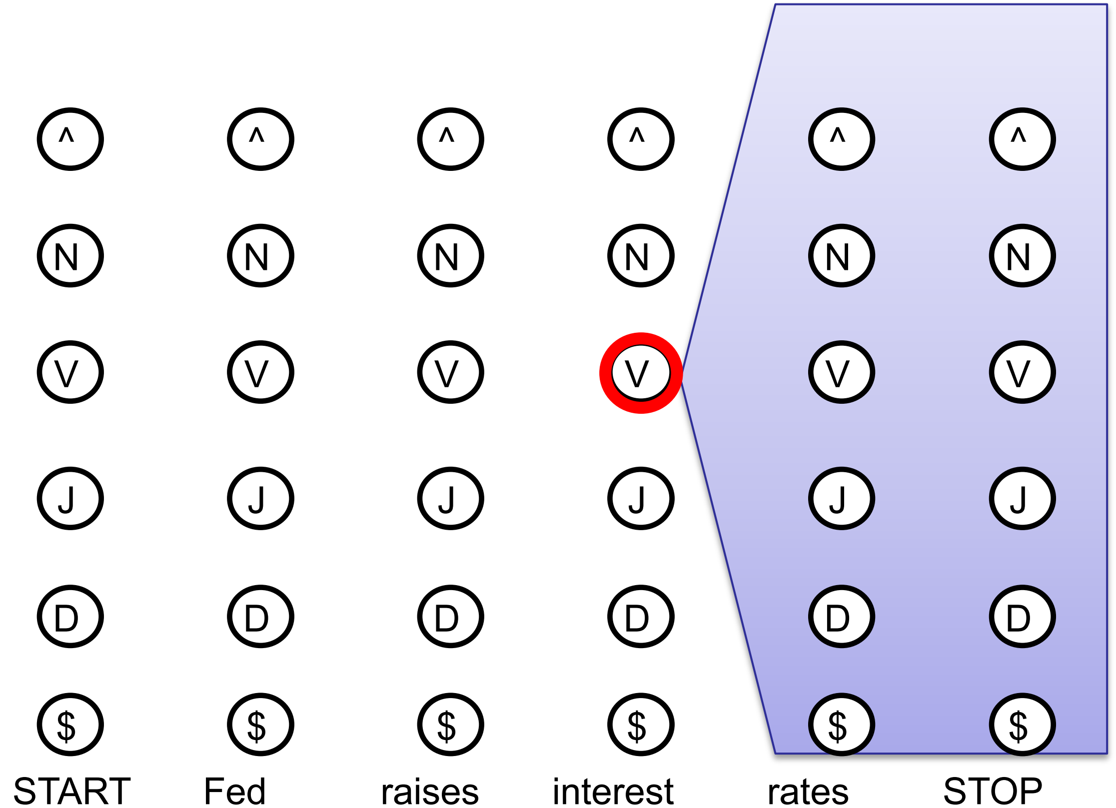
$$= \sum_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \alpha(i-1, y_{i-1})$$



The State Lattice / Trellis: Backward

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

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



Forward Backward 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:

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

    - For i = n-1 ... 0

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

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

  $O(|\mathcal{K}|)$ time to compute each entry
- Total runtime: $O(n|\mathcal{K}|^2)$
- Q: How does this compare to Viterbi?
- A: Exactly the same!!!

Other Marginal Inference

- We've been doing this:

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

- Can we compute this?

$$p(x_1 \ldots x_n) = \sum_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

$$= \quad \ldots ? \ldots \quad p(x_1 \ldots x_n, y_i)$$

$$= \sum_{y_i} p(x_1 \ldots x_n, y_i)$$

Other Marginal Inference

- Can we compute this?

$$p(x_1 ... x_n) = \sum_{y_i} p(x_1 ... x_n, y_i)$$

- Relation with forward quantity?

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

$$p(x_1 ... x_n) = \sum_{y_1 \cdots y_n} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

$$= \quad ...?... \quad \alpha(n, y_n)$$

$$= \sum_{y_n} q(STOP|y_n)\alpha(n, y_n) := \alpha(n+1, STOP)$$

Unsupervised Learning (EM) Intuition

- We've been doing this:

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

- What we really want is this: (which we now know how to compute!)

$$p(y_i|x_1 \ldots x_n) = \frac{p(x_1 \ldots x_n, y_i)}{p(x_1 \ldots x_n)}$$

- This means we can compute the expected count of things

$$\text{(expected) count(NN)} = \sum_i p(y_i = \text{NN}|x_1 \ldots x_n)$$

Unsupervised Learning (EM) Intuition

- What we really want is this: (which we now know how to compute!)

$$p(y_i|x_1 \ldots x_n) = \frac{p(x_1 \ldots x_n, y_i)}{p(x_1 \ldots x_n)}$$

- This means we can compute the expected count of things:

$$\text{(expected) count(NN)} = \sum_i p(y_i = \text{NN}|x_1 \ldots x_n)$$

- If we have this: $p(y_i y_{i+1}|x_1 \ldots x_n) = \dfrac{p(x_1 \ldots x_n, y_i, y_{i+1})}{p(x_1 \ldots x_n)}$

- We can also compute expected transition counts:

$$\text{(expected) count(NN} \rightarrow \text{VB)} = \sum_i p(y_i = \text{NN}, y_{i+1} = \text{VB}|x_1 \ldots x_n)$$

- Above marginals can be computed as

$$p(x_1 \ldots x_n, y_i) = \alpha(i, y_i)\beta(i, y_i)$$

$$p(x_1 \ldots x_n, y_i, y_{i+1}) = \alpha(i, y_i)q(y_{i+1}|y_i)e(x_{i+1}|y_{i+1})\beta(i+1, y_{i+1})$$

Unsupervised Learning (EM) Intuition

- Expected emission counts:

$$
\begin{aligned}
\text{(expected) count}(\text{NN} \rightarrow \text{apple}) &= \sum_i p(y_i = \text{NN}, x_i = \text{apple} | x_1 \ldots x_n) \\
&= \sum_{i:x_i=\text{apple}} p(y_i = \text{NN} | x_1 \ldots x_n)
\end{aligned}
$$

- Maximum Likelihood Parameters (Supervised Learning):

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

- For Unsupervised Learning, replace the actual counts with the expected counts.

Expectation Maximization

- Initialize transition and emission parameters
  - Random, uniform, or more informed initialization
- Iterate until convergence
  - **E-Step:**
    - Compute expected counts

$$\text{(expected) count(NN)} = \sum_i p(y_i = \text{NN} | x_1 ... x_n)$$

$$\text{(expected) count(NN} \rightarrow \text{VB)} = \sum_i p(y_i = \text{NN}, y_{i+1} = \text{VB} | x_1 ... x_n)$$

$$\text{(expected) count(NN} \rightarrow \text{apple)} = \sum_i p(y_i = \text{NN}, x_i = \text{apple} | x_1 ... x_n)$$

  - **M-Step:**
    - Compute new transition and emission parameters (using the expected counts computed above)

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

- Convergence? Yes. Global optimum? No

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** *HMM=(A,B)*

initialize A and B
iterate until convergence
E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \ \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\,a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \ \forall\, t,\ i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{j=1}^{N} \xi_t(i,j)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

Equivalent to the procedure given in the textbook (J&M) – slightly different notations

How is Unsupervised Learning Possible (at all)?

- I water the garden everyday
- Saw a weird bug in that garden …
- While I was thinking of an equation …

Noun

S: (n) **garden** (a plot of ground where plants are cultivated)

S: (n) **garden** (the flowers or vegetables or fruits or herbs that are cultivated in a garden)

S: (n) **garden** (a yard or lawn adjoining a house)

Verb

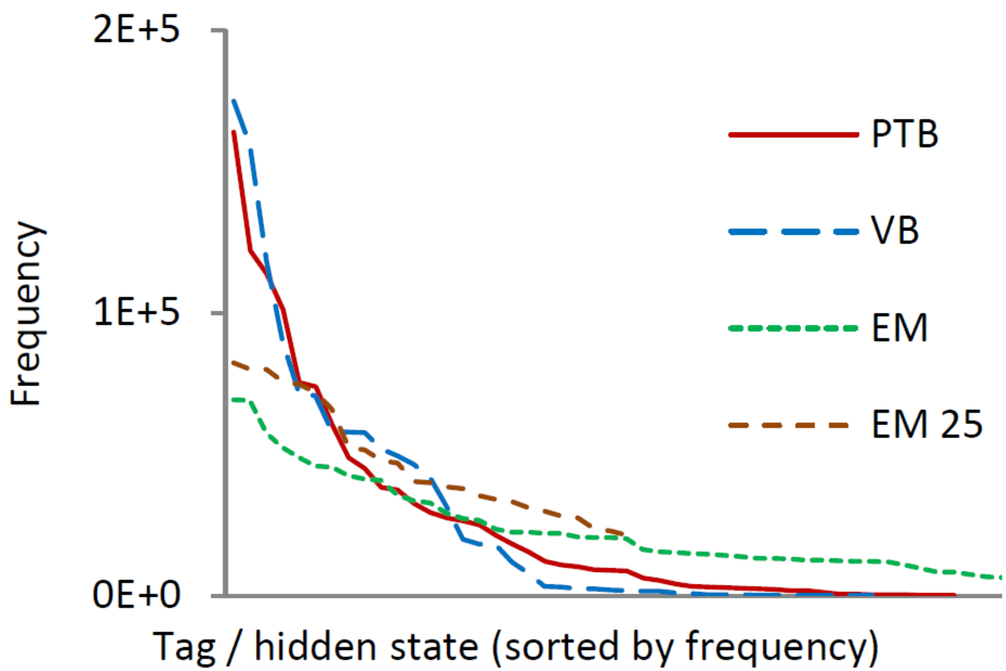
S: (v) **garden** (work in the garden) *"My hobby is gardening"*

Adjective

S: (adj) **garden** (the usual or familiar type) *"it is a common or garden sparrow"*

Does EM learn good HMM POS-taggers?

- *"Why doesn't EM find good HMM POS-taggers"*, Johnson, EMNLP 2007



HMMs estimated by EM generally assign a roughly equal number of word tokens to each hidden state, while the empirical distribution of tokens to POS tags is highly skewed

Unsupervised Learning Results

- EM for HMM
  - POS Accuracy: 74.7%
- Bayesian HMM Learning [Goldwater, Griffiths 07]
  - Significant effort in specifying prior distriubtions
  - Integrate our parameters e(x|y) and t(y'|y)
  - POS Accuracy: 86.8%
- Unsupervised, feature rich models [Smith, Eisner 05]
  - Challenge: represent p(x,y) as a log-linear model, which requires normalizing over all possible sentences x
  - Smith presents a very clever approximation, based on local neighborhoods of x
  - POS Accuracy: 90.1%
- Newer, feature rich methods do better, not near supervised SOTA

Quiz: p(S1) vs. p(S2)

- S1 = Colorless green ideas sleep furiously.
- S2 = Furiously sleep ideas green colorless
  - *"It is fair to assume that neither sentence (S1) nor (S2) had ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be ruled out on identical grounds as equally "remote" from English"* (Chomsky 1957)
- How would p(S1) and p(S2) compare based on (smoothed) bigram language models?
- How would p(S1) and p(S2) compare based on marginal probability based on POS-tagging HMMs?
  - i.e., marginalized over all possible sequences of POS tags