CMP462: Natural Language Processing

Lecture 5: Hidden Markov Models

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2014

Agenda

- Tagging Problems
- Supervised Learning
- Markov Models
- Hidden Markov Models
  - Definition
  - Training
  - Inference
- Trigram HMM for POS Tagging

Acknowledgment: Most slides adapted from Michael Collins' NLP class on Coursera, from Dan Klein, and from Chris Manning's NLP class on Coursera.

Tagging Problems: Parts of Speech

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street,

as their CEO Alan Mulally announced first quarter results.

OUTPUT:

Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV topping/V

forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N

Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.

N = Noun

V = Verb

P = Preposition

Adv = Adverb

Adj = Adjective

Tagging: Named Entity Recognition

INPUT: Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced rst quarter results.

OUTPUT: Profits soared at [Company Boeing Co.], easily topping forecasts on [Location Wall Street], as their CEO [Person Alan Mulally] announced first quarter results.

Tagging: Named Entity Recognition

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT:

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

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

Tagging

- We saw how to solve this using supervised machine learning techniques
- In particular, we saw how to train and use Maximum Entropy Models for NER
- MaxEnt Models are a type of *discriminative* models
- Here we will discuss Hidden Markov Models, a type of *generative* models

Our Goal

Given the training set:

1 Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./.

2 Mr./NNP Vinken/NNP is/VBZ chairman/NN of/IN Elsevier/NNP N.V./NNP ,/, the/DT Dutch/NNP publishing/VBG group/NN ./.

3 Rudolph/NNP Agnew/NNP ,/, 55/CD years/NNS old/JJ and/CC chairman/NN of/IN Consolidated/NNP Gold/NNP Fields/NNP PLC/NNP ,/, was/VBD named/VBN a/DT nonexecutive/JJ director/NN of/IN this/DT British/JJ industrial/JJ conglomerate/NN ./.

…

38,219 It/PRP is/VBZ also/RB pulling/VBG 20/CD people/NNS out/IN of/IN Puerto/NNP Rico/NNP ,/, who/WP were/VBD helping/VBG Huricane/NNP Hugo/NNP victims/NNS ,/, and/CC sending/VBG them/PRP to/TO San/NNP Francisco/NNP instead/RB ./.

We want to induce a function/algorithm that maps new sentences to their tag sequences.

Supervised Learning

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \ldots n$.
- Each $x^{(i)}$ is an input and each $y^{(i)}$ is a label
- Task is to learn a function f mapping inputs to labels $f(x)$.
- Discriminative Models
  - Learn the distribution $p(y \mid x)$ from the training examples
  - Define $f(x) = \arg\max_y p(y \mid x)$
  - Example: Maximum Entropy Models

Supervised Learning

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \ldots n$.
- Each $x^{(i)}$ is an input and each $y^{(i)}$ is a label
- Task is to learn a function f mapping inputs to labels $f(x)$.
- Generative Models
  - Learn the distribution $p(x, y)$ from the training examples
  - Often we have $p(x, y) = p(y)\, p(x \mid y)$
  - Define $f(x) = \arg\max_y p(y \mid x) = \arg\max_y \frac{p(y)\, p(x \mid y)}{p(x)}$ $= \arg\max_y p(y)\, p(x \mid y)$
  - Example: Naive Bayes Models

Markov Sequences

- Consider a sequence of random variables $X_1, X_2, \ldots X_m$
- Each variable X_i takes only values in $\{1, \ldots, k\}$
- These can be, for example, the POS tags (labels) for a sentence
- How do we model the joint distribution:

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m)$$

Markov Assumption

- A simplifying assumption that makes modeling the joint distribution of the sequence much easier
- Assume that each variable is independent of anything else *conditioned* on its predecessor(s)

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m)$$

The Chain Rule

$$= P(X_1 = x_1) \prod_{j=2}^{m} P(X_j = x_j | X_1 = x_1, \ldots, X_{j-1} = x_{j-1})$$

The Markov Assumption

$$= P(X_1 = x_1) \prod_{j=2}^{m} P(X_j = x_j | X_{j-1} = x_{j-1})$$

Assume that X_j is independent of anything else given X_{j-1}

Homogeneous Markov Assumption

- One more assumption that further simplifies the model
- Assume that the *transition probabilities* (the probability of a state given its predecessor) are independent of time (the index j)
- This gives us:

$$P(X_j = x_j | X_{j-1} = x_{j-1}) = q(x_j | x_{j-1})$$

- Where $q(x' | x)$ is some function
- For example, in POS tagging, each x_j is a tag (e.g. V or N), and $q(N | V)$ gives us the probability that a Noun follows a Verb irrespective of their position

Markov Models

- So our model is:

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m)$$
$$= P(x_1, \ldots, x_m) = q(x_1) \prod_{j=2}^{m} q(x_j | x_{j-1})$$

- The parameters in the model:

$$q(x) \text{ for } x \in \{1, \ldots, k\}$$
$$\text{s.t. } q(x) \geq 0 \text{ and } \sum_{x=1}^{k} q(x) = 1$$

$$q(x' | x) \text{ for } x, x' \in \{1, \ldots, k\}$$
$$\text{s.t. } q(x' | x) \geq 0 \text{ and } \sum_{x'=1}^{k} q(x' | x) = 1$$

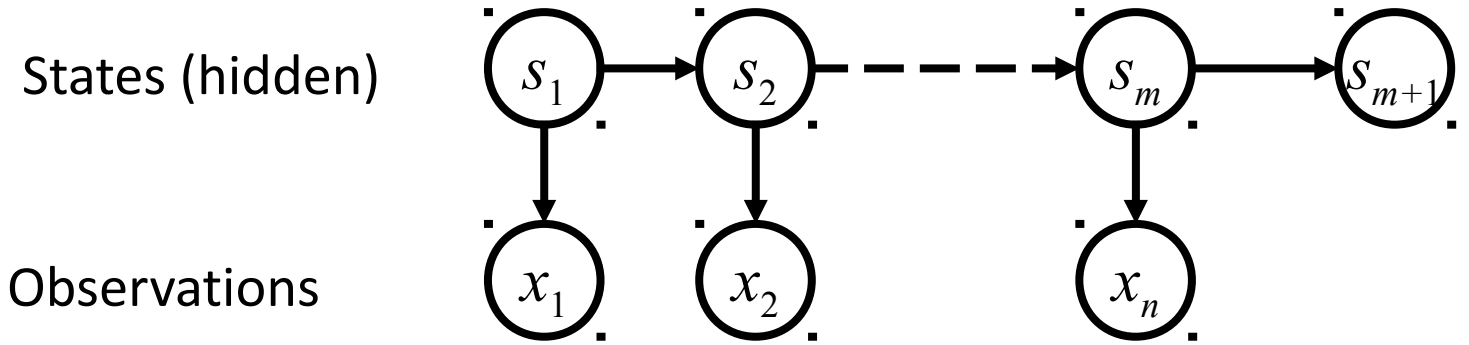
Generative Markov Models

- To generate a Markov sequence:
  - Pick a random x_1 from the distribution $q(x)$
  - For $j = 2 \ldots m$
    - Choose x_j at random from the distribution $q(x_j \mid x_{j-1})$
- For example: "*the dog laughs*"
  - Pick a POS tag at random for the first word from the prior distribution of the POS tags e.g. DET (determiner)
  - For each word starting at the second, pick a POS tag at random from the distribution *given* the previous tag
    - For *dog*: pick a tag at random given that the previous tag is DET e.g. N
    - For *laughs*: pick a tag at random given that the previous tag is N e.g. V

Pairs of Sequences

- In many cases we need to model *pairs* of sequences together
- The problem is that we only *observe* one sequence but we are also interested in the *hidden* sequence
- Examples:
  - POS tagging: We observe the sequence of words (the sentence) but we are also interested in the POS tags/labels
  - NER: We observe the sequence of words but we are also interested in the NER tags.

Hidden Markov Models



- We have two sequences: $S_1, S_2, \ldots S_m$ states (hidden) and $X_1, X_2, \ldots X_m$ observations (visible)
- Each state *generates* the corresponding observation e.g. S_j generates X_j and each state *depends* on its predecessor
- We want to model the joint distribution:

$$P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m)$$

Assumptions in HMM I

- The state sequence forms a Markov Chain i.e. state transition probabilities depend only on the previous state

$$P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m)$$

$$= P(S_1 = s_1, \ldots, S_m = s_m) \times P(X_1 = x_1, \ldots, X_m = x_m | S_1 = s_1, \ldots, S_m = s_m)$$

Chain Rule

$$= t(s_1) \prod_{j=2}^{m} t(s_j | s_{j-1}) \times P(X_1 = x_1, \ldots, X_m = x_m | S_1 = s_1, \ldots, S_m = s_m)$$

State sequence is a Markov Chain

Assumptions in HMM II

- Each observation depends only on the underlying state

$$P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m)$$

$$= t(s_1) \prod_{j=2}^{m} t(s_j | s_{j-1}) \times$$
$$P(X_1 = x_1, \ldots, X_m = x_m | S_1 = s_1, \ldots, S_m = s_m)$$

$$= t(s_1) \prod_{j=2}^{m} t(s_j | s_{j-1}) \times$$ Chain Rule
$$\prod_{j=1}^{m} P(X_j = x_j | S_1 = s_1, \ldots, S_m = s_m, X_1 = x_1, \ldots, X_{j-1} = x_{j-1})$$

Observations depend on state only

$$= t(s_1) \prod_{j=2}^{m} t(s_j | s_{j-1}) \times \prod_{j=1}^{m} e(x_j | s_j)$$

Generative HMM

- We can generate a sequence pair $x_1, \ldots x_m$ and $s_1, \ldots s_m$ as follows:
  - Pick s_1 at random from the distribution $t(s)$. Pick x_1 from the distribution $e(x \mid s_1)$.
  - For $j = 2 \ldots m$:
    - Choose s_j at random from the distribution $t(s \mid s_{j-1})$
    - Choose x_j at random from the distribution $e(x \mid s_j)$

$$P(x_1, \ldots, x_m, s_1, \ldots, s_m) = t(s_1) \prod_{j=2}^{m} t(s_j \mid s_{j-1}) \prod_{j=1}^{m} e(x_j \mid s_j)$$

HMM Parameters

- The model is:

$$P(x_1, \ldots, x_m, s_1, \ldots, s_m) = t(s_1) \prod_{j=2}^{m} t(s_j | s_{j-1}) \prod_{j=1}^{m} e(x_j | s_j)$$

- The parameters:

1. Initial State Parameters: $t(s)$ for $s \in \{1, \ldots, k\}$

2. Transition Parameters: $t(s' | s)$ for $s, s' \in \{1, \ldots, k\}$

3. Emission Parameters: $e(x | s)$ for $s, s' \in \{1, \ldots, k\}$ and $x \in \{1, \ldots, \sigma\}$

- How do we get these parameters?

Parameter Estimation

- Assume we have n training examples each containing a sentence $x_1, \ldots x_m$ and its corresponding state sequence $s_1, \ldots s_m$
- Define:

$$[[\pi]] = 1 \text{ if } \pi \text{ is true, and 0 otherwise}$$

Parameter Estimation: Transition Parameters

- Assume we have n training examples each containing a sentence $x_1, \ldots x_m$ and its corresponding state $s_1, \ldots s_m$
- Define the number of times state s is followed by state s' in training example i as:

$$\text{count}(i, s \rightarrow s') = \sum_{j=1}^{m} [[s_{i,j} = s \text{ and } s_{i,j+1} = s']]$$

- The maximum likelihood estimates of the *transition probabilities* are then:

$$t(s'|s) = \frac{\sum_{i=1}^{n} \text{count}(i, s \rightarrow s')}{\sum_{i=1}^{n} \sum_{s''} \text{count}(i, s \rightarrow s'')}$$

Parameter Estimation: Transition Parameters

$$t(s'|s) = \frac{\sum_{i=1}^{n} \text{count}(i, s \rightarrow s')}{\sum_{i=1}^{n} \sum_{s''} \text{count}(i, s \rightarrow s'')}$$

- For example, to estimate $t(\text{V}|\text{ADJ})$ we count how many times in the training set a Verb came after an Adjective and divide by the number of adjectives in the training set.

$$t(\text{V}|\text{ADJ}) = \frac{\sum_{i=1}^{n} \text{count}(i, \text{ADJ} \rightarrow \text{V})}{\sum_{i=1}^{n} \sum_{s''} \text{count}(i, \text{ADJ} \rightarrow s'')}$$

Parameter Estimation: Emission Parameters

- Assume we have n training examples each containing a sentence $x_1, \ldots x_m$ and its corresponding state $s_1, \ldots s_m$
- Define the number of times state s is paired with emission (observable) x in training example i as:

$$\text{count}(i, s \rightarrow x) = \sum_{j=1}^{m} [[s_{i,j} = s \text{ and } x_{i,j} = x]]$$

- The maximum likelihood estimates of the *emission probabilities* are then:

$$e(x|s) = \frac{\sum_{i=1}^{n} \text{count}(i, s \rightarrow x)}{\sum_{i=1}^{n} \sum_{x'} \text{count}(i, s \rightarrow x')}$$

Parameter Estimation: Emission Parameters

$$e(x|s) = \frac{\sum_{i=1}^{n} \text{count}(i, s \rightarrow x)}{\sum_{i=1}^{n} \sum_{x'} \text{count}(i, s \rightarrow x')}$$

- For example: to estimate $e(\text{bright}|\text{ADJ})$ we count in the training set how many times bright was tagged as an adjective ADJ and divide by the number of adjectives i.e.

$$e(bright|\text{ADJ}) = \frac{\sum_{i=1}^{n} \text{count}(i, \text{ADJ} \rightarrow bright)}{\sum_{i=1}^{n} \sum_{x'} \text{count}(i, \text{ADJ} \rightarrow x')}$$

Parameter Estimation: Initial State Parameters

- Assume we have n training examples each containing a sentence $x_1, \ldots x_m$ and its corresponding state $s_1, \ldots s_m$
- Define whether state s is the start of the sequence in training example i as:

$$\text{count}(i, s) = [[s_{i,1} = s]]$$

- The maximum likelihood estimates of the *initial state probabilities* are then:

$$t(s) = \frac{\sum_{i=1}^{n} \text{count}(i, s)}{n}$$

Parameter Estimation: Initial State Parameters

$$t(s) = \frac{\sum_{i=1}^{n} \text{count}(i, s)}{n}$$

- For example: to estimate $t(\text{N})$ i.e. the probability of starting a sentence with a Noun, we count in the training set how many times a Noun started a sentence and divide by the number of sentences

$$t(\text{N}) = \frac{\sum_{i=1}^{n} \text{count}(i, \text{N})}{n}$$

HMM Inference

- Now we have a training set, and we used it to estimate the HMM parameters
- How do we use these parameters to tag an sequence?
- This is called *inference*.
- In particular, for any observed sequence $x_1, \ldots x_m$, we want to find the sequence of states $s_1, \ldots s_m$ that maximizes their joint probability i.e. find

$$\text{argmax}_{s_1,\ldots,s_m} P(x_1,\ldots,x_m,s_1,\ldots,s_m)$$

$$P(x_1,\ldots,x_m,s_1,\ldots,s_m) = t(s_1)\prod_{j=2}^{m} t(s_j|s_{j-1})\prod_{j=1}^{m} e(x_j|s_j)$$

HMM Inference

- The *brute force* solution is hopelessly inefficient:
  - For all possible state sequences, compute the joint probability
  - Return the sequence of states with the maximum probability
- How many possible sequences are there?
  - k^n
- We will use instead the *Viterbi Algorithm*, a dynamic programming solution that finds the optimal sequence in *polynomial* time instead of *exponential*

The Viterbi Algorithm

- Define $\pi[j, s]$ as the *maximum* probability for any state sequence that ends with state s at position j.
- More formally:

$$j=1 \quad \rightarrow \quad \pi[1,s]=t(s)e(x_1|s)$$

$$j>1 \quad \rightarrow \quad \pi[j,s]=\max_{s_1,\ldots,s_{j-1}} t(s_1)e(x_1|s_1) \times \prod_{l=2}^{j-1} t(s_l|s_{l-1})e(x_l|s_l) \times \underbrace{t(s|s_{j-1})e(x_j|s)}$$

The sequence ends with state s

- To compute $\pi[j, s]$ naively, we need to check k^{j-1} possible sequences, but we will do it recursively

The Viterbi Algorithm

- Assume we know $\pi[j, s]$ for all states s at a given position j.
- To extend the sequence one more step i.e. to compute $\pi[j+1, s']$ for some state s', we can reuse our computations at position j , more formally:

$$\pi[j+1, s'] = \max_s \left(\pi[j, s] \, t(s'|s) \, e(x_{j+1}|s') \right)$$

Optimal solution at step j ending with state s

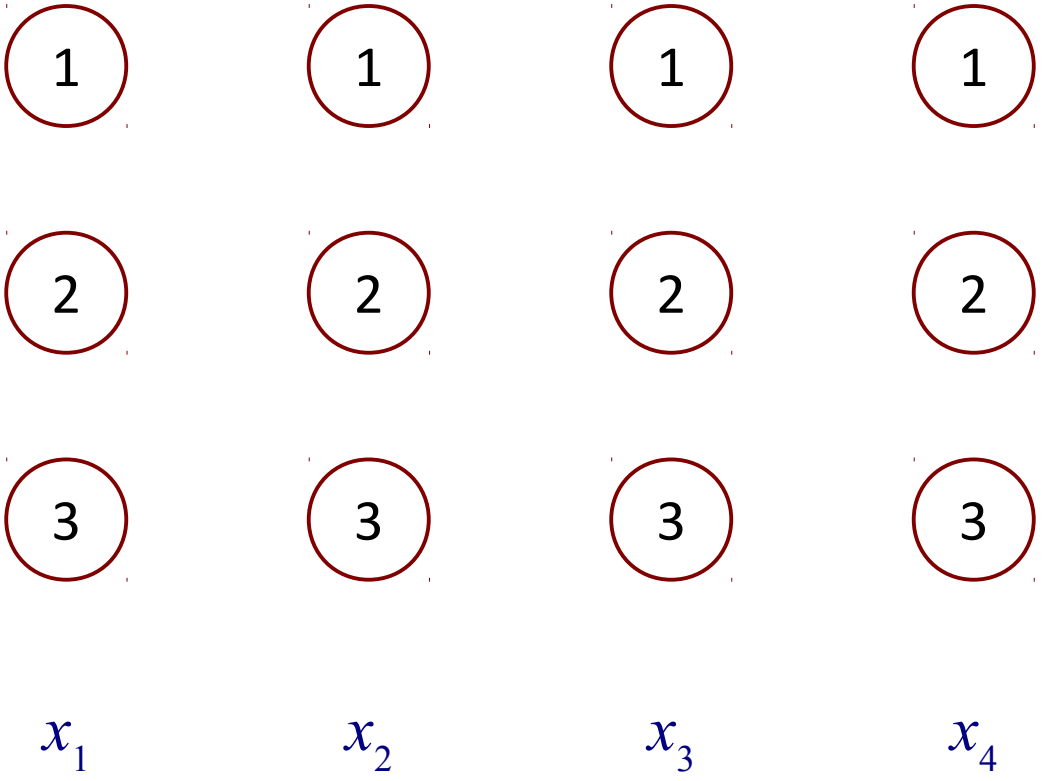
Contribution of adding state s' after state s

- The optimal solution will then be:

$$\max_{s_1, \ldots, s_m} P(x_1, \ldots, x_m, s_1, \ldots, s_m) = max_s \, \pi[m, s]$$

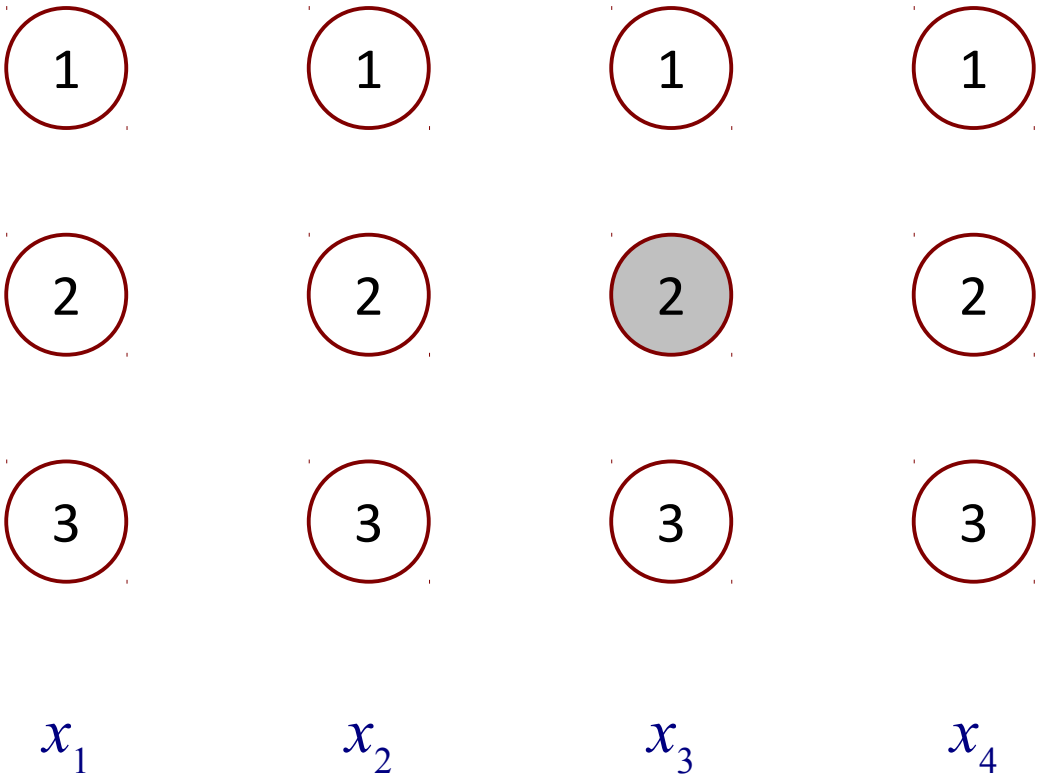
The State Lattice

$m = 4$ observations and $k = 3$ possible states



The State Lattice

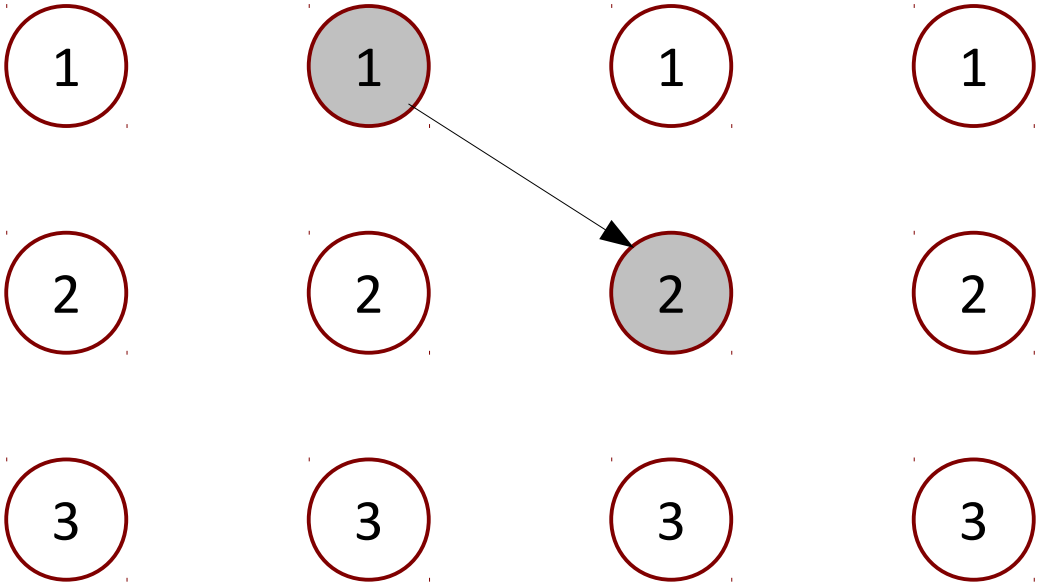
$m = 4$ observations and $k = 3$ possible states



- Assume we want to compute $\pi[3, 2]$ at position $j = 3$
- We already computed $\pi[2, s]$ for $s = \{1, 2, 3\}$
- The previous state can be any of states $s = 1, 2, \text{or } 3$, but which one?
- We try each one and pick the best!

The State Lattice

$m = 4$ observations and $k = 3$ possible states



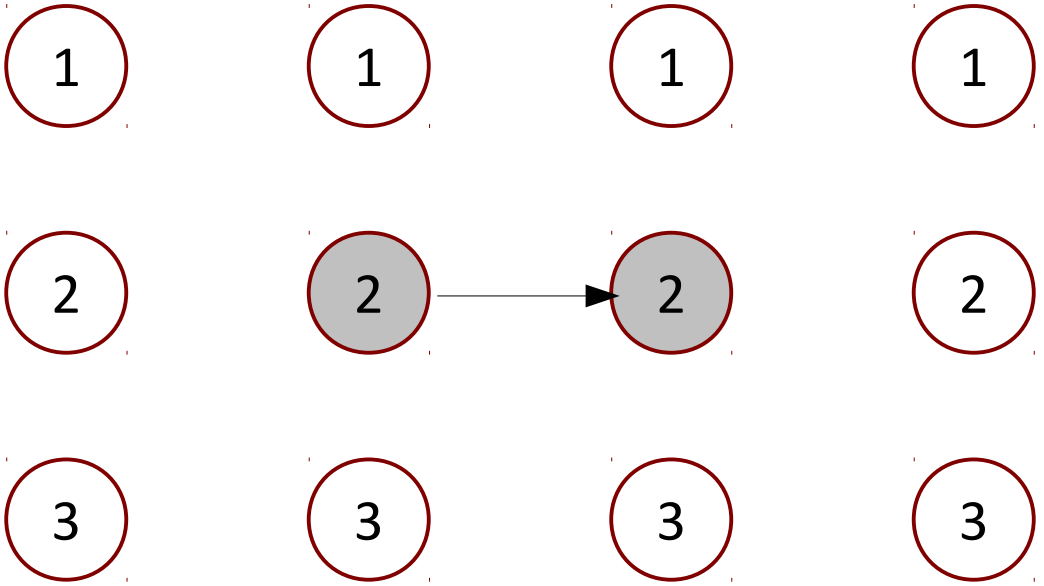
$$\pi[3,2] = \pi[2,1]\, t(2|1)\, e(x_3|2)$$

Optimal solution at step 2 ending with state 1

Contribution of adding state 2 after state 1

The State Lattice

$m = 4$ observations and $k = 3$ possible states



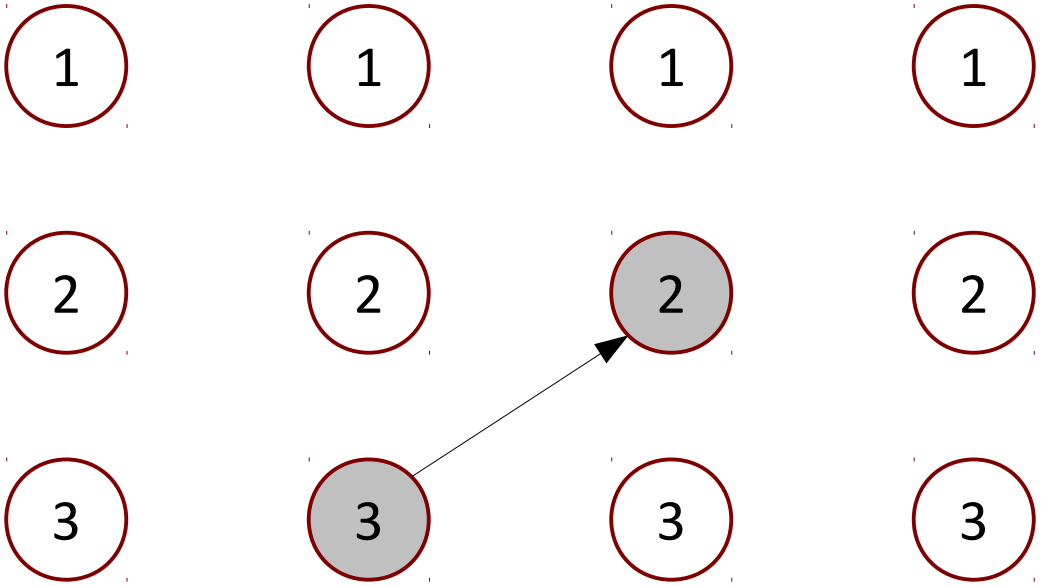
$$\pi[3,2] = \pi[2,2]\, t(2|2)\, e(x_3|2)$$

Optimal solution at step 2 ending with state 2

Contribution of adding state 2 after state 2

The State Lattice

$m = 4$ observations and $k = 3$ possible states



$$\pi[3,2] = \pi[2,3]\, t(2|3)\, e(x_3|2)$$

Optimal solution at step 2 ending with state 3

Contribution of adding state 2 after state 3

The Viterbi Algorithm

- Initialization: for $s = 1 \ldots k$:

$$\pi[1, s] = t(s)\, e(x_1 | s)$$

- For $j = 2 \ldots m,\ s = 1 \ldots k$:

$$\pi[j, s] = \max_{s'} \left(\pi[j-1, s']\, t(s | s')\, e(x_j | s) \right)$$

- The optimal solution is then:

$$\max_{s_1, \ldots, s_m} P(x_1, \ldots, x_m, s_1, \ldots, s_m) = \max_s \pi[m, s]$$

- The algorithm runs in time $O(m\, k^2)$:
  - We need to compute $\pi[j, s]$ for all j and s i.e. $O(m\, k)$
  - For each one, we need to check k other previous states

The Viterbi Algorithm

- Initialization: for $s = 1 \ldots k$:

$$\pi[1, s] = t(s)e(x_1|s)$$

- For $j = 2 \ldots m,\ s = 1 \ldots k$:

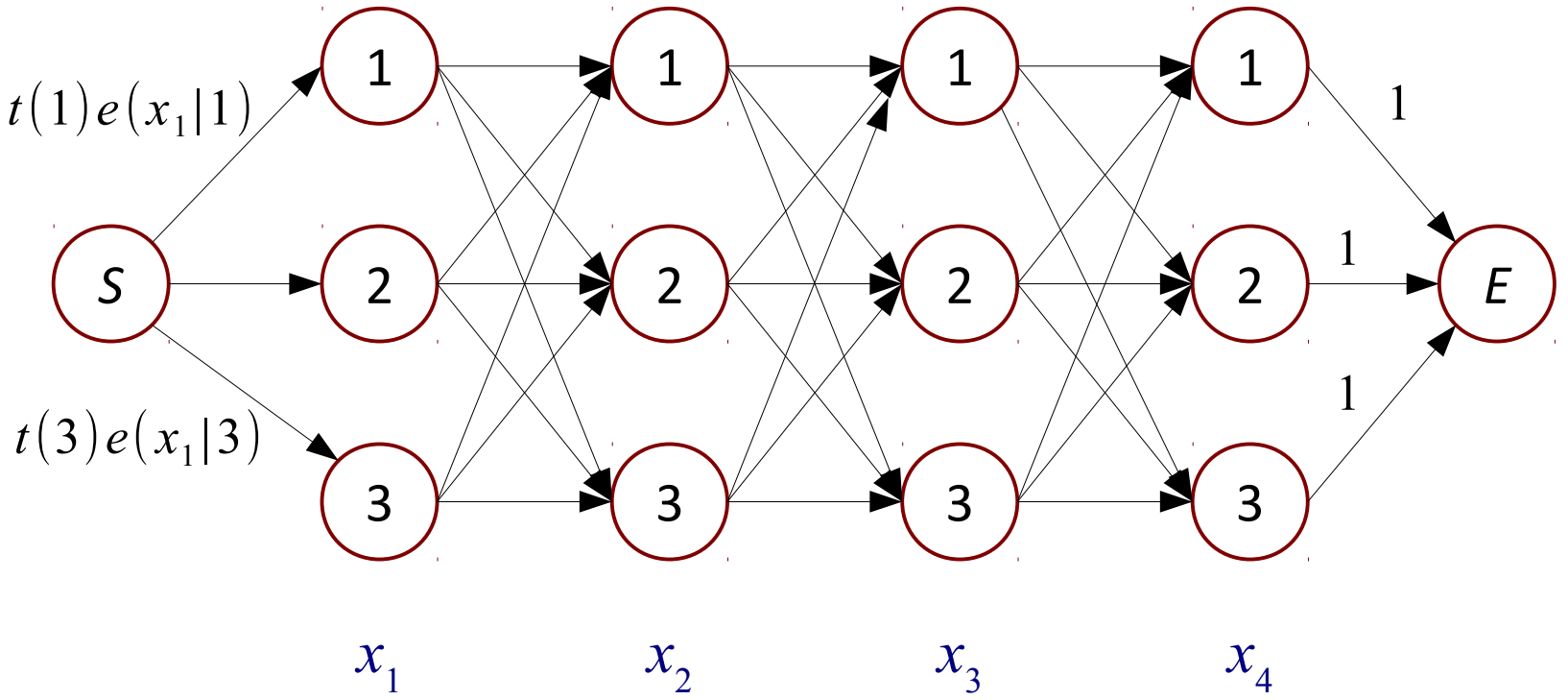
$$\pi[j, s] = \max_{s'}\left(\pi[j-1, s']t(s|s')e(x_j|s)\right)$$

$$bp[j, s] = \arg\max_{s'}\left(\pi[j-1, s']t(s|s')e(x_j|s)\right)$$

- The *back pointer* $bp[j, s]$ stores the best previous state to reach state s and allows us to reconstruct the best sequence.

Viterbi as a "Shortest-Path" Algorithm

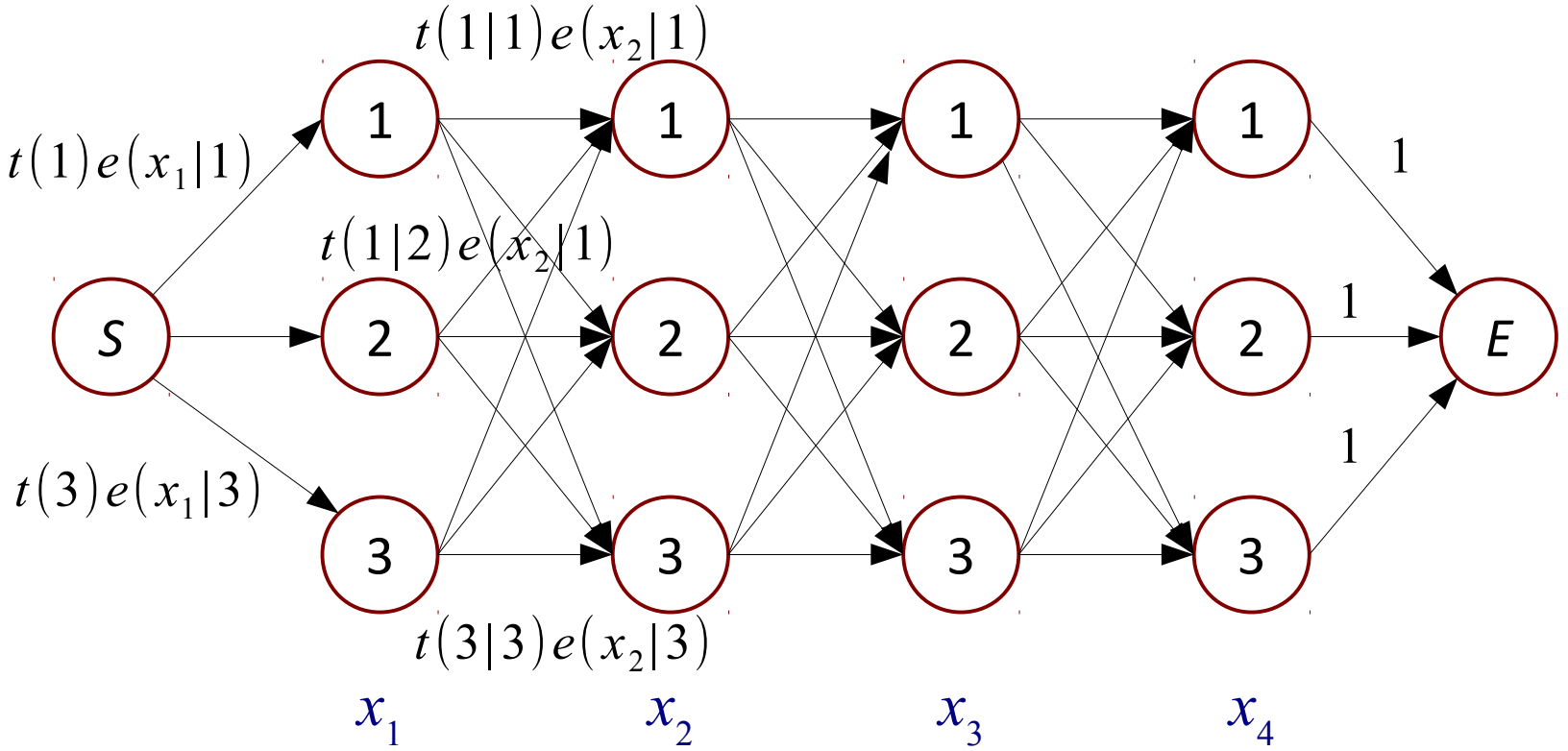
$m = 4$ observations and $k = 3$ possible states



- Define the weight of the path as the product of the weights on the path
- Add a *Source* vertex S with edges to vertices $(1, s)$ with weight $t(s)\, e(x_1 \mid s)$
- Add an End vertex E with edges from vertices (m, s) with weight 1

Viterbi as a "Shortest-Path" Algorithm

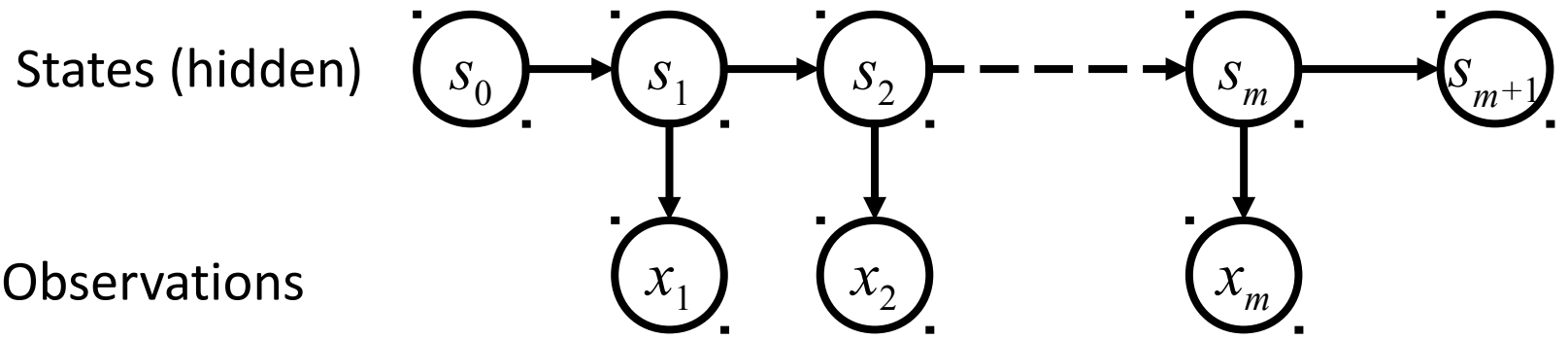
$m = 4$ observations and $k = 3$ possible states



- Add edges from each vertex (j, s) to each vertex $(j+1, s')$ with weight $t(s' \mid s)\, e(x_{j+1} \mid s')$
- Find the path of maximum weight from the starting vertex S to end vertex E

HMM for Tagging

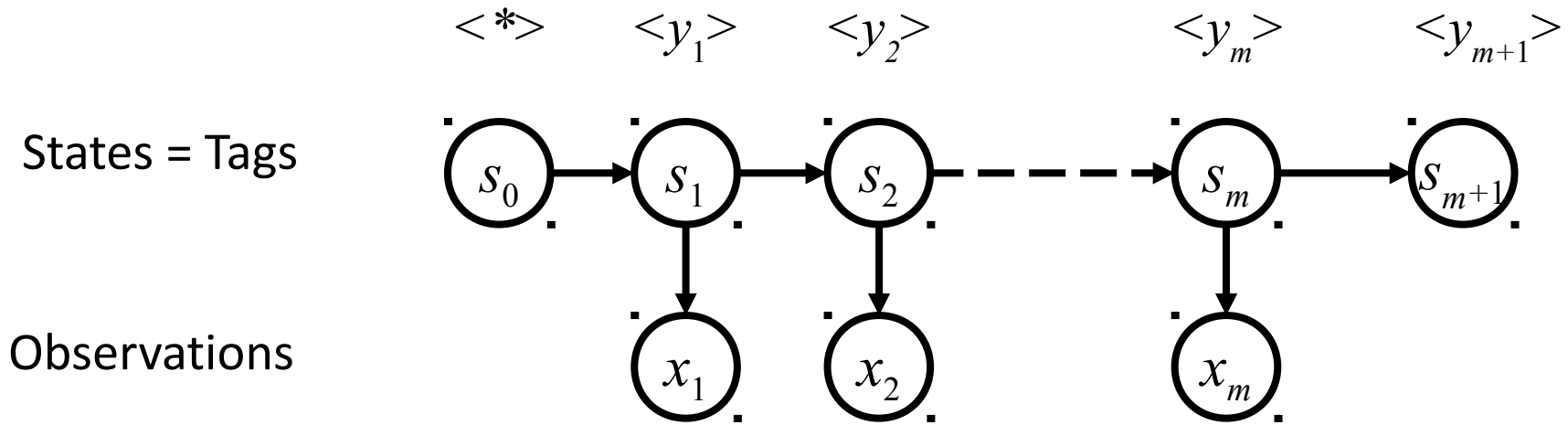
$$P(x_1, \ldots, x_m, s_0, s_1, \ldots, s_m, s_{m+1}) = t(s_0) \prod_{j=1}^{m+1} t(s_j | s_{j-1}) \prod_{j=1}^{m} e(x_j | s_j)$$



- The states are tag n-grams e.g. $s_i = \mathrm{N}$ or $s_i = \langle \mathrm{N}, \mathrm{V} \rangle$
- The states encode everything necessary about the past
- Usually has a dedicated start and end states s_0 and s_{m+1}
- Words (observations) depend only on the current state
- States follow a Markov model

HMM for Tagging

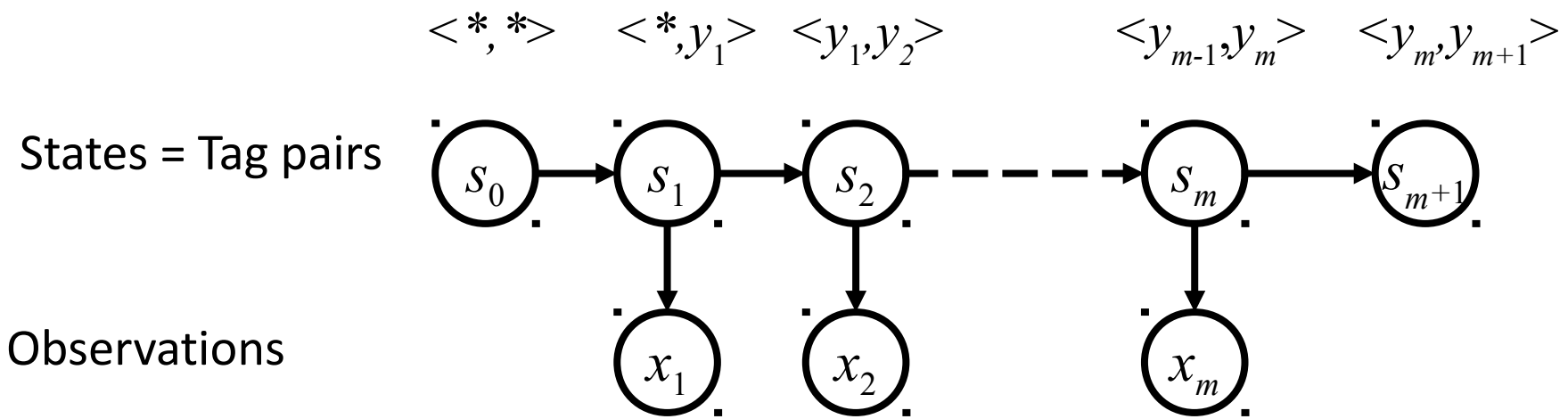
$$P(x_1, \ldots, x_m, s_0, s_1, \ldots, s_m, s_{m+1}) = t(s_0) \prod_{j=1}^{m+1} t(s_j | s_{j-1}) \prod_{j=1}^{m} e(x_j | s_j)$$



Bigram HMM Tagger

HMM for Tagging

$$P(x_1, \ldots, x_m, s_0, s_1, \ldots, s_m, s_{m+1}) = t(s_0) \prod_{j=1}^{m+1} t(s_j | s_{j-1}) \prod_{j=1}^{m} e(x_j | s_j)$$



Trigram HMM Tagger

HMM for Tagging

- We will focus now on a particular model, namely Trigram HMM for POS Tagging
- In particular, we will review the following in the context of POS tagging:
  - Definition: how to define the model
  - Training: how to estimate the parameters
  - Inference: how to use the model to find the best tag sequence for a given sentence

Hidden Markov Models (HMM)

- We have an input sequence $x = x_1 \ldots x_m$ (x_j is the j^{th} word in the sentence)
- We have a tag sequence $y = y_1 \ldots y_m$ (y_j is the j^{th} tag in the sequence)
- We will use a Hidden Markov Model to define

$$p(x_1, x_2, \ldots x_m, y_1, y_2, \ldots y_m)$$

for any sentence $x_1 \ldots x_m$ and tag sequence $y_1 \ldots y_m$

- The most likely tag sequence for any sentence x is:

$$\arg\max_{y1, \ldots, ym} p(x_1, x_2, \ldots x_m, y_1, y_2, \ldots y_m)$$

Trigram HMMs

- For any sentence $x = x_1 \ldots x_m$ and any tag sequence $y = y_1 \ldots y_{m+1}$ where y_{m+1} = STOP, we define the joint probability distribution of the sentence and tag sequence as:

$$p(x_1, \ldots, x_m, y_1, \ldots, y_{m+1}) = \prod_{j=1}^{m+1} t(y_j | y_{j-2}, y_{j-1}) \prod_{j=1}^{m} e(x_j | y_j)$$

where we assumed that $y_0 = y_{-1}$ = * (a special padding symbol)

- Note: we can define the state as bi-grams of POS tags as explained before *or* we can modify the transition probabilities to rely on the *previous two states* as we did here.

Trigram HMMs

- For any sentence $x = x_1 \ldots x_m$ and any tag sequence $y = y_1 \ldots y_{m+1}$ where y_{m+1} = STOP, we define the joint probability distribution of the sentence and tag sequence as:

$$p(x_1, \ldots, x_m, y_1, \ldots, y_{m+1}) = \prod_{j=1}^{m+1} t(y_j | y_{j-2}, y_{j-1}) \prod_{j=1}^{m} e(x_j | y_j)$$

where we assumed that $y_0 = y_{-1} = *$

- Parameters of the model:
  - Transition Probabilities: $t(s \mid u, v)$ for any tags triplet s, u, v
  - Emission Probabilities: $e(x \mid s)$ for any tag s and word x
  - Note: Initial State Probabilities are included in the transition probabilities with $t(s \mid *, *)$

Example

- If we have $m = 3$ and $x = the\ dog\ laughs$ and the tag sequence $y = \mathrm{D\ N\ V\ STOP}$, then

$$p(x_1, \ldots, x_m, y_1, \ldots, y_{m+1}) = t(\mathrm{D}|*,*)\, t(\mathrm{N}|*,\mathrm{D}) \times t(\mathrm{V}|\mathrm{D},\mathrm{N})\, t(\mathrm{STOP}|\mathrm{N},\mathrm{V}) \times e(\mathrm{the}\,|\,\mathrm{D})\, e(\mathrm{dog}\,|\,\mathrm{N})\, e(\mathrm{laughs}\,|\,\mathrm{V})$$

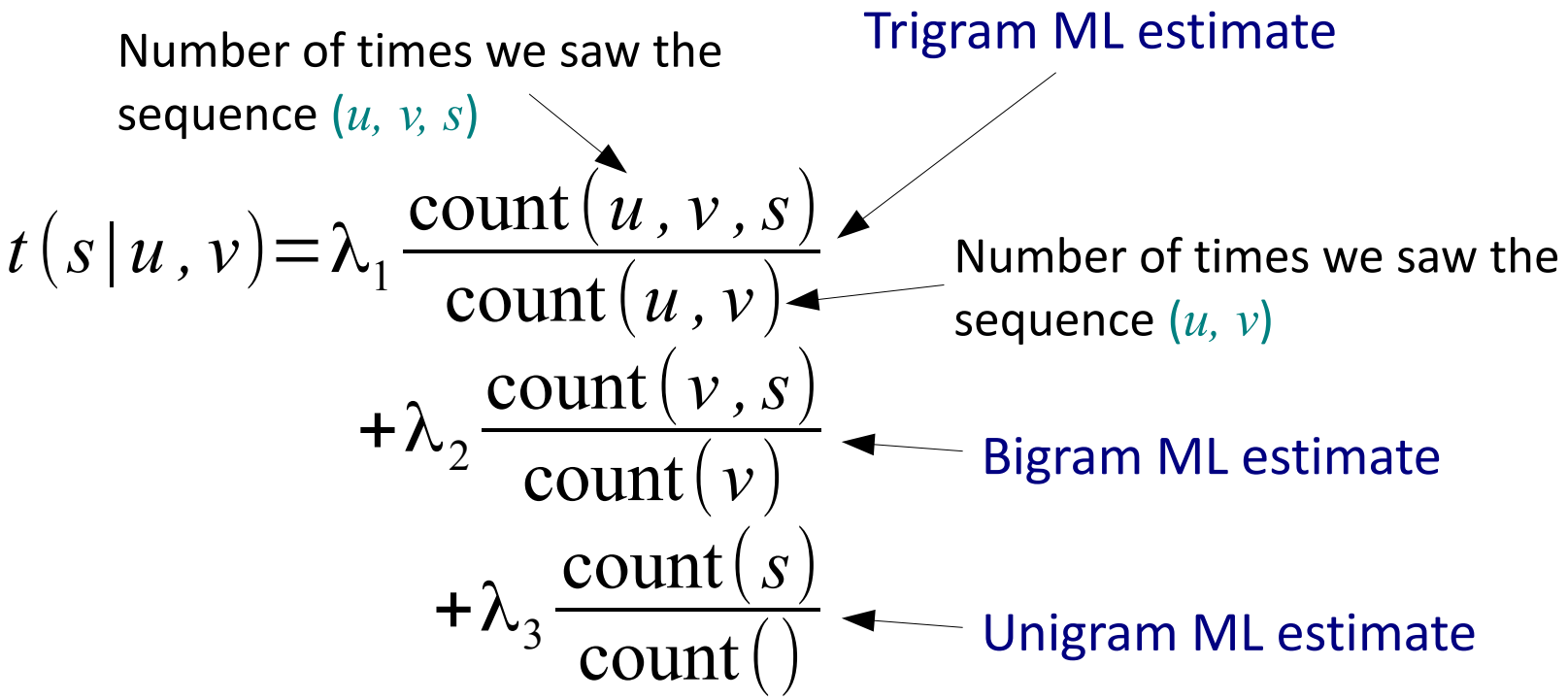
$$p(x_1, \ldots, x_m, y_1, \ldots, y_{m+1}) = \prod_{j=1}^{m+1} t(y_j | y_{j-2}, y_{j-1}) \prod_{j=1}^{m} e(x_j | y_j)$$

Smoothed Parameter Estimation

- We can define *maximum likelihood* estimates as we did earlier
- Instead, we will define *smoothed* estimates to alleviate the zero count problem i.e. when some counts are zero (unseen in the training set)
- We will use the *interpolation* method

Smoothed Parameter Estimation

- We estimate $t(s \mid u, v)$ as:



$$t(s \mid u, v) = \lambda_1 \frac{\text{count}(u, v, s)}{\text{count}(u, v)} + \lambda_2 \frac{\text{count}(v, s)}{\text{count}(v)} + \lambda_3 \frac{\text{count}(s)}{\text{count}()}$$

$$\sum_i \lambda_i = 1 \text{ and } \lambda_i \geq 0$$

Smoothed Parameter Estimation

- We estimate $e(x \mid s)$ as:

$$e(x|s) = \frac{\text{count}(s, x)}{\text{count}(s)}$$

Number of times we saw the word x with the tag s

Number of times we saw the tag s

- Now what about words that don't appear in the training set?
  - Their probabilities will be zero!

$$e(x|s) = 0 \text{ for all tags } s$$

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = 0 \quad !!$$

Low Frequency Words

- Step 1: Split the vocabulary into two sets
  - Frequent words: words occurring ≥ 5 times during training
  - Low frequency words: all other words
- Step 2: Map low frequency words into a small finite set, depending on prefixes, suffixes, word shape, … etc.
- We do this for both the training and test sets, then perform the parameter estimation process

Low Frequency Words: Example

[Bikel et. al 1999] **(named-entity recognition)**

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

Step 2: Map the low frequency words into word classes

Low Frequency Words: 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

Viterbi Algorithm for Trigram HMM

- Define $\pi[j, u, v]$ as the *maximum* probability for any state sequence that ends with states u and v at position j.
- More formally:

$$\pi[j, u, v] = \max_{y_{-1}, y_0, y_1, \ldots, y_j : y_{k-1}=u, y_k=v} \prod_{k=1}^{j} t(y_k | y_{k-2}, y_{k-1}) e(x_k | y_k)$$

The sequence ends with states u, v

- Define S to be the set of available tags and S_j for $j = -1, 0, \ldots, m$ to be the set of possible tags at position j:

$$S_{-1} = S_0 = \{*\}$$
$$S_j = S \text{ for } j \in \{1 \ldots m\}$$

Example

1	2	3	4	5	6	7	8
The	man	saw	the	dog	with	the	telescope
*	*	D	D	D	D	D	P	D	
		N	N	N	N	N			Tags
		V	V	V	V	V			
		P	P	P	P	P			

- Focus on computing $\pi[7, P, D]$ where P is Preposition, and D is Determiner
- Assume the possible tags are $S = \{D, N, V, P\}$
- We have an exponential number of tag sequences that end in $P\ D$ at position 7
- $\pi[7, P, D]$ is the maximum probability of all these possible sequences

Viterbi Algorithm for Trigram HMM

- Assume we know $\pi[j-1, w, u]$ for all states w, u at a given position $j-1$.
- To extend the sequence one more step i.e. to compute $\pi[j, u, v]$ for some state v, we can reuse our computations at position $j-1$. More formally, for any j in $\{1 \ldots m\}$ and any u in S_{j-1} and v in S_j, we have:

$$\pi[j, u, v] = \max_{w \in S_{j-2}} \left(\underbrace{\pi[j-1, w, u]}_{\text{Optimal solution at step } j \text{ ending with states } w, u} \underbrace{t(v|w,u)e(x_j|v)}_{\text{Contribution of adding state } v \text{ after states } w, u} \right)$$

- Base case:

$$\pi[0, *, *] = 1$$

Example

1	2	3	4	5	6	7	8
The	man	saw	the	dog	with	the	telescope
*	*	D	D	D	D	D	P	D
		N	N	N	N	N		
		V	V	V	V	V		
		P	P	P	P	P		

Tags

- Focus on computing $\pi[7, \mathrm{P}, \mathrm{D}]$
- $S_5 = S = \{\mathrm{D}, \mathrm{N}, \mathrm{V}, \mathrm{P}\}$

$$\pi[7, P, D] = \max_{w \in S_5} \left(\pi[6, w, P] \, t(D \mid w, P) \, e(\text{the} \mid D) \right)$$

Viterbi Algorithm for Trigram HMM

- Initialization: $\pi[0,*,*]=1$
- Define: $S_{-1}=S_0=\{*\}$ and $S_j=S$ for $j\in\{1\ldots m\}$
- For $j = 1 \ldots m$, $u \in S_{j-1}$, $v \in S_j$:

$$\pi[j,u,v]=\max_{w\in S_{j-2}}\left(\pi[j-1,w,u]t(v|w,u)e(x_j|v)\right)$$

$$bp[j,u,v]=\arg\max_{w\in S_{j-2}}\left(\pi[j-1,w,u]t(v|w,u)e(x_j|v)\right)$$

- Return: $\max_{u\in S_{m-1},v\in S_m}\left(\pi[m,u,v]t(STOP|u,v)\right)$
- The *back pointer* $bp[j, u, v]$ stores the best previous state w to reach state u, v and allows us to reconstruct the best sequence.

Viterbi Algorithm for Trigram HMM

- Initialization: $\pi[0,*,*]=1$
- Define: $S_{-1}=S_0=\{*\}$ and $S_j=S$ for $j\in\{1\ldots m\}$
- For $j = 1 \ldots m$, $u \in S_{j-1}$, $v \in S_j$:

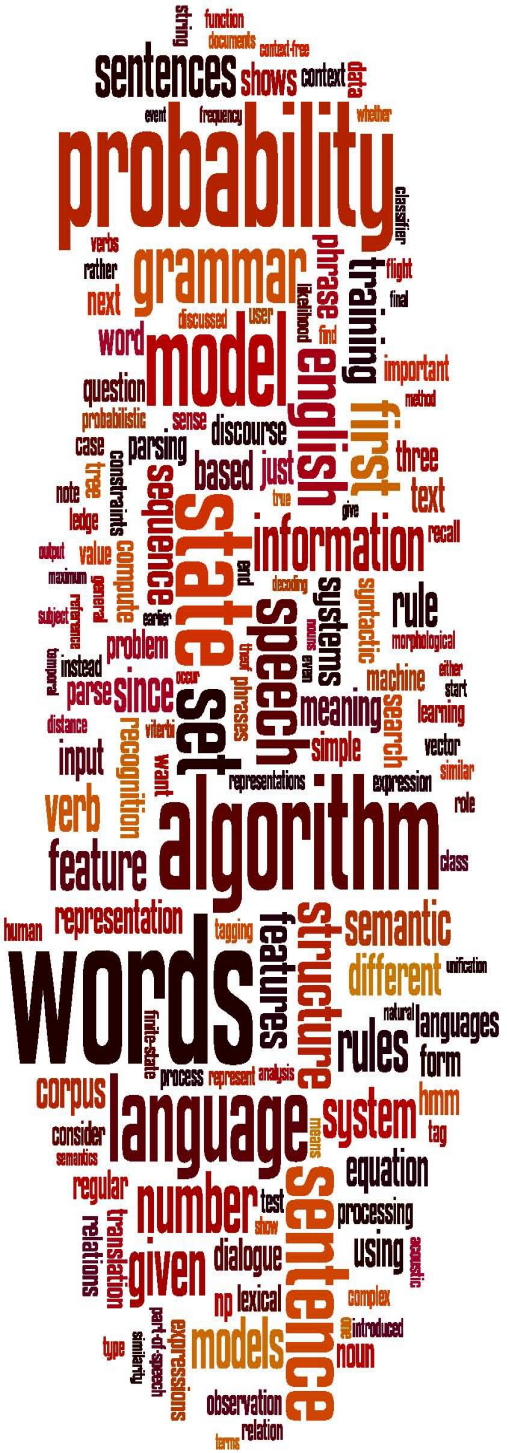
$$\pi[j,u,v]=\max_{w\in S_{j-2}}\left(\pi[j-1,w,u]t(v|w,u)e(x_j|v)\right)$$

$$bp[j,u,v]=\arg\max_{w\in S_{j-2}}\left(\pi[j-1,w,u]t(v|w,u)e(x_j|v)\right)$$

- Return: $\max_{u\in S_{m-1},v\in S_m}\left(\pi[m,u,v]t(STOP|u,v)\right)$
- Set: $(y_{m-1},y_m)=\arg\max_{u\in S_{m-1},v\in S_m}\left(\pi[m,u,v]t(STOP|u,v)\right)$
- For $j = m–2 \ldots 1$: $y_j=bp[j+2,y_{j+1},y_{j+2}]$

Running Time

- $O(m\ |S|^3)$
  - We need to calculate $\pi[j, u, v]$ for all j, u, v i.e. $O(m\ |S|^2)$
  - For each such calculation, we need to loop over all possible tags i.e. total is $O(m\ |S|^3)$
- Notice that before we said that it takes $O(m\ k^2)$, but k was the number of *states* not the number of *tags*.
- In fact, in the Trigram HMM, the number of possible states is $|S|^2$ if we use the state as a bigram of tags
- But, we don't search over all possible states as in the normal HMM. Why?
- Because the two consecutive states have to *match*.



Sequence Models for Part-of-speech tagging

Christopher Manning

Sources of information

- What are the main sources of information for POS tagging?
  - Knowledge of neighboring words
    - Bill saw that man yesterday
    - NNP NN DT NN NN
    - VB VB(D) IN VB NN
  - Knowledge of word probabilities
    - *man* is rarely used as a verb….
- The latter proves the most useful, but the former also helps

More and Better Features ➔ Feature-based tagger

- Can do surprisingly well just looking at a word by itself:
  - Word: the: the → DT
  - Lowercased word: Importantly: importantly → RB
  - Prefixes: unfathomable: un- → JJ
  - Suffixes: Importantly: -ly → RB
  - Capitalization: Meridian: CAP → NNP
  - Word shapes: 35-year: d-x → JJ

- Then build a maxent (or whatever) model to predict tag
  - Maxent P(t|w) (without any context information, just looking at the words) 93.7% overall / 82.6% for unknown words

Overview: POS Tagging Accuracies

- Rough accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - Maxent P(t|w): 93.7% / 82.6%
  - TnT (HMM++): 96.2% / 86.0%
  - MEMM tagger: 96.9% / 86.9%
  - Bidirectional dependencies: 97.2% / 90.0%
  - Upper bound: ~98% (human agreement)

Most errors on unknown words

How to improve supervised results?

- Build better features by looking at the kinds of errors our classifiers make!

		RB					
PRP	VBD	IN	RB	IN	PRP	VBD	.
They	left	as	soon	as	he	arrived	.

  - We could fix this with a feature that looked at the next word

	JJ				
NNP	NNS	VBD		VBN	.
Intrinsic	flaws	remained		undetected	.

  - We could fix this by linking capitalized words (Intrinsic) to their lowercase versions (intrinsic)

Recap

- Tagging Problems
- Supervised Learning
- Markov Models
- Hidden Markov Models
  - Definition
  - Training
  - Inference
- Trigram HMM for POS Tagging