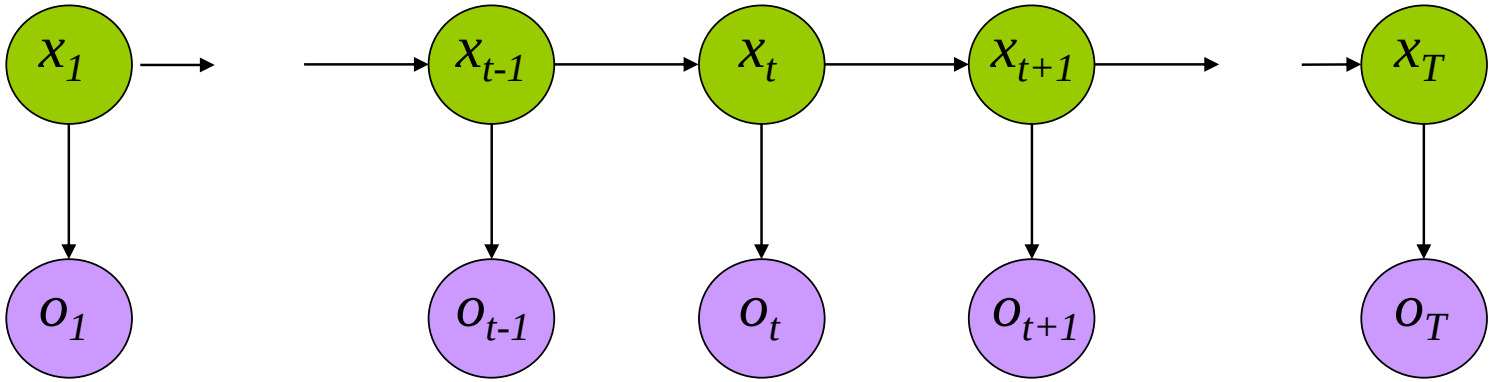
Content

Decoding Solution



$$P(O \mid \mu) = \sum_{i=1}^{N} \alpha_i(T)$$ **Forward Procedure**

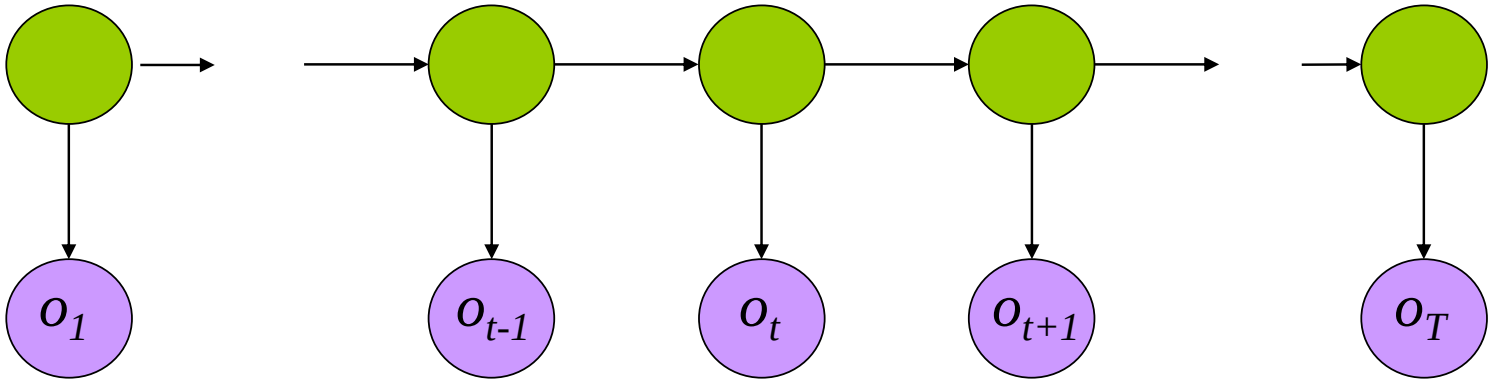
$$P(O \mid \mu) = \sum_{i=1}^{N} \pi_i \beta_i(1)$$ **Backward Procedure**

$$P(O \mid \mu) = \sum_{i=1}^{N} \alpha_i(t)\beta_i(t)$$ **Combination**

David Meir Blei, Hidden Markov Models, 1999 (Slides)

Content

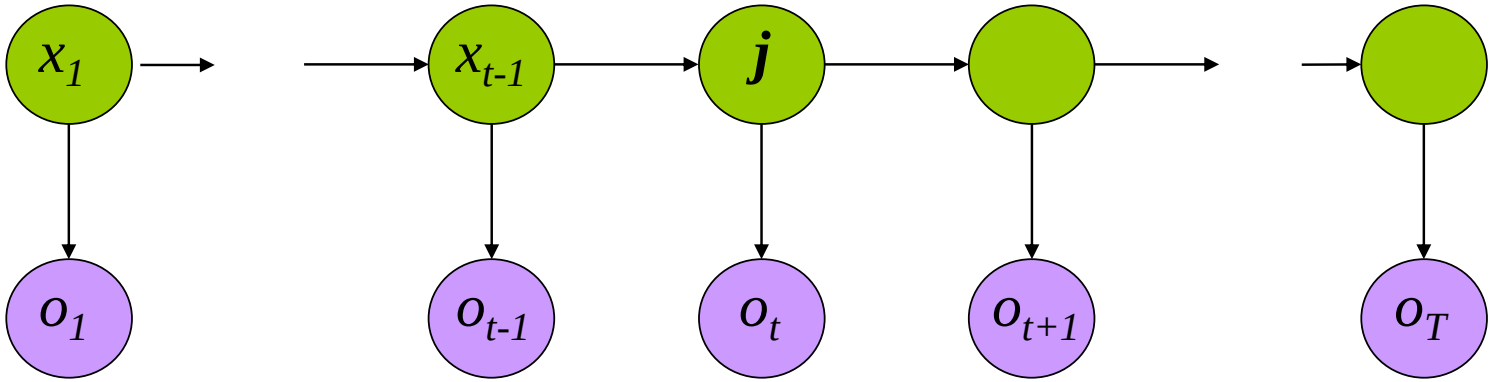
Best State Sequence



- Find the state sequence that best explains the observations
- **Viterbi** algorithm
- $\arg\max_X P(X \mid O)$

David Meir Blei, Hidden Markov Models, 1999 (Slides)

Viterbi Algorithm

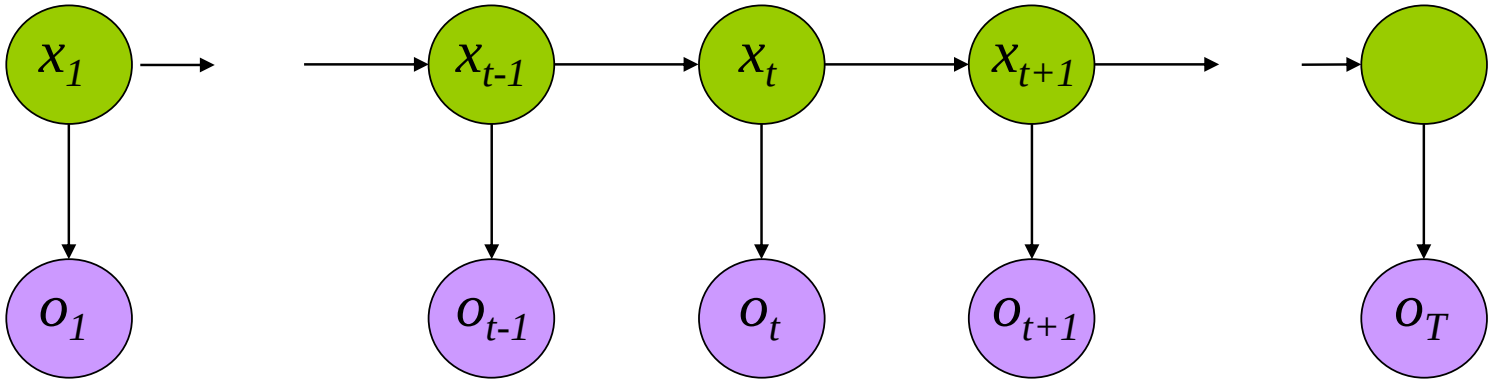


$$\delta_j(t) = \max_{x_1 \ldots x_{t-1}} P(x_1 \ldots x_{t-1}, o_1 \ldots o_{t-1}, x_t = j, o_t)$$

The state sequence which maximizes the probability of seeing the observations to time t-1, landing in state j, and seeing the observation at time t

David Meir Blei, Hidden Markov Models, 1999 (Slides)

Viterbi Algorithm



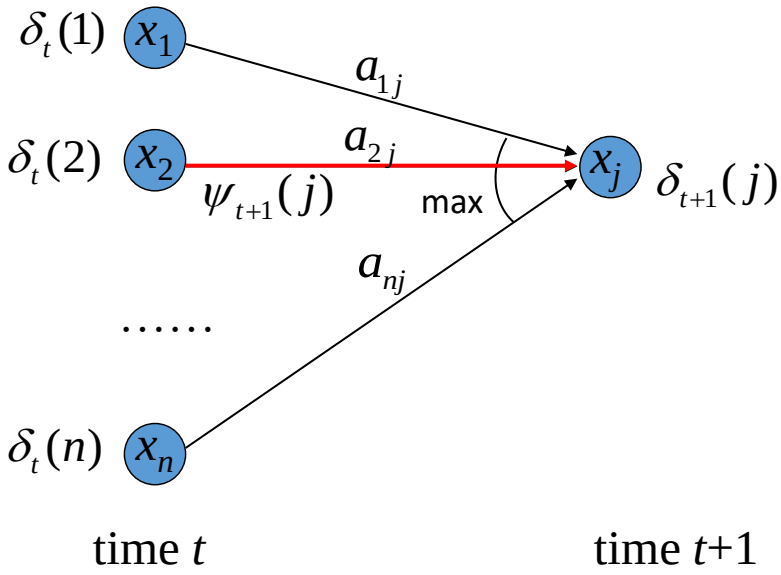
$$\delta_j(t) = \max_{x_1 \ldots x_{t-1}} P(x_1 \ldots x_{t-1}, o_1 \ldots o_{t-1}, x_t = j, o_t)$$

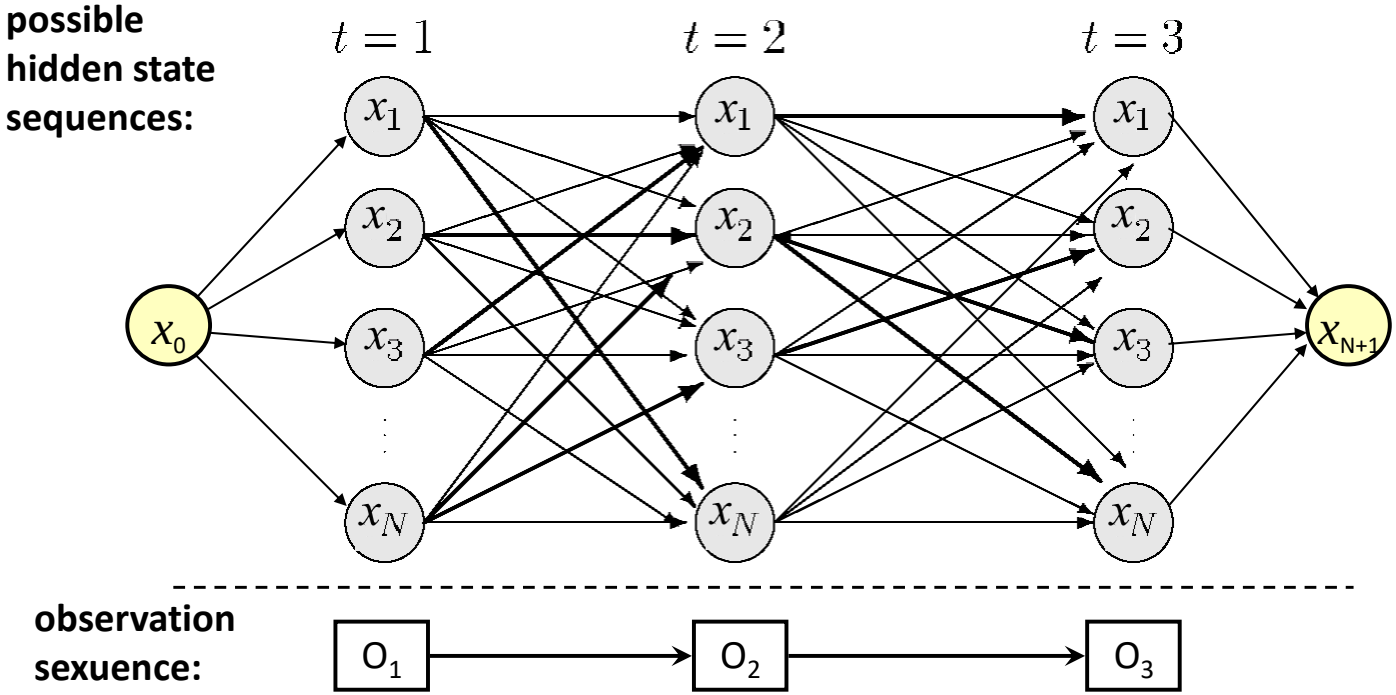
$$\delta_j(t+1) = \max_i \delta_i(t) a_{ij} b_{jo_{t+1}}$$

$$\psi_j(t+1) = \arg\max_i \delta_i(t) a_{ij} b_{jo_{t+1}}$$

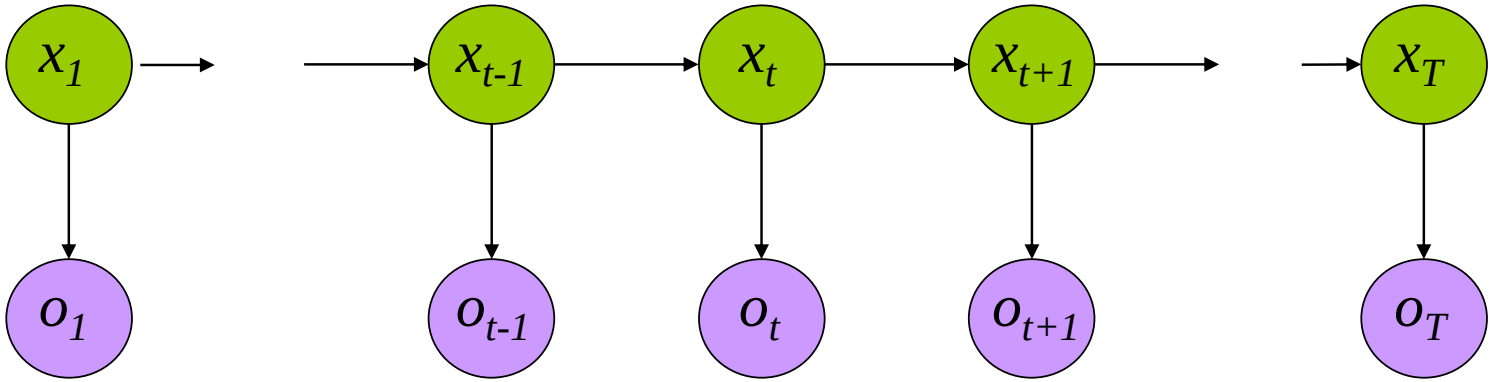
Recursive Computation

David Meir Blei, Hidden Markov Models, 1999 (Slides)


$\delta_t(1)$
x_1
a_{1j}
$\delta_t(2)$
x_2
a_{2j}
x_j
$\delta_{t+1}(j)$
$\psi_{t+1}(j)$
max
a_{nj}
.....
$\delta_t(n)$
x_n
time t
time t+1


possible
hidden state
sequences:
$t = 1$
$t = 2$
$t = 3$
x_1
x_2
x_3
x_N
x_0
x_{N+1}
observation
sexuence:
O_1
O_2
O_3

Viterbi Algorithm



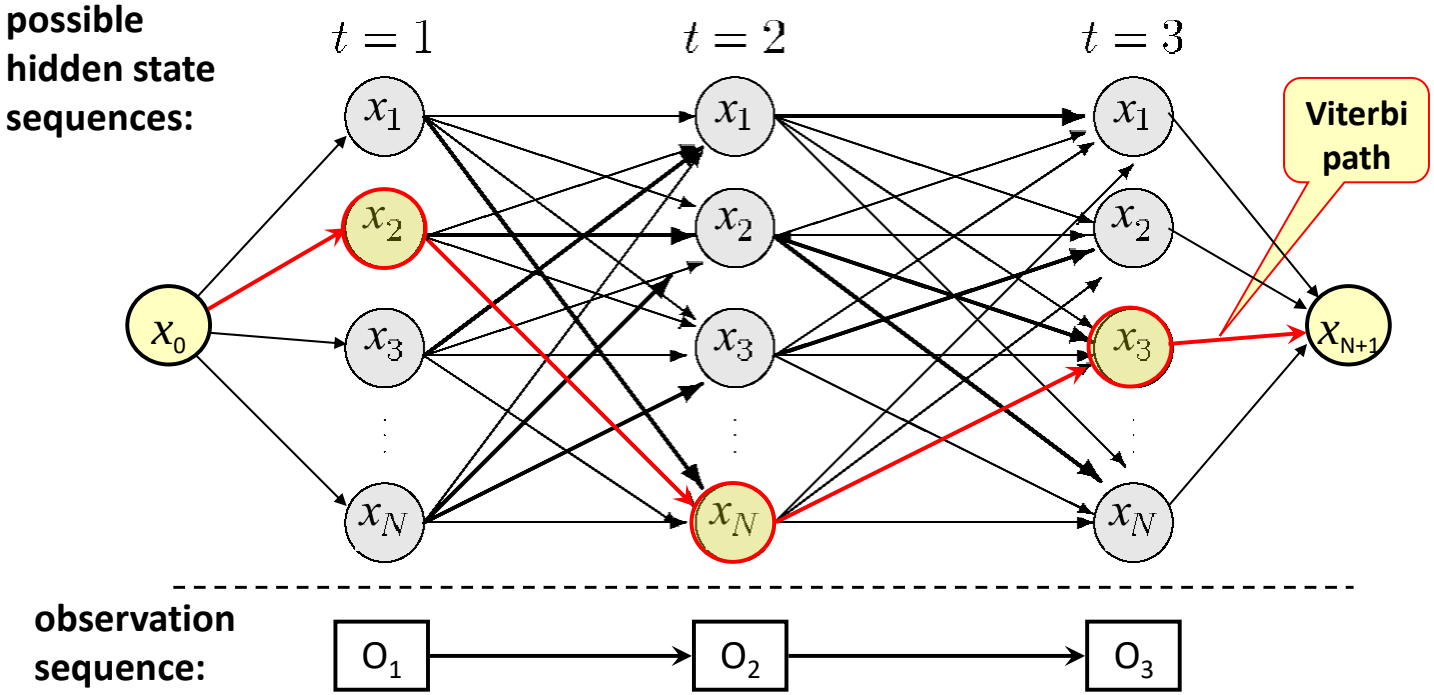
$$\hat{X}_T = \arg\max_i \delta_i(T)$$

$$\hat{X}_t = \psi_{\hat{X}_{t+1}}(t+1)$$

$$P(\hat{X}) = \arg\max_i \delta_i(T)$$

Compute the most likely state sequence by working backwards

David Meir Blei, Hidden Markov Models, 1999 (Slides)



# of possible hidden state sequences: N^T

Time complexity of Viterbi algorithm: O(N2T)

Viterbi Algorithm

- Finding the best single sequence means computing $\text{argmax}_Q P(Q|O,\lambda)$, equivalent to $\text{argmax}_Q P(Q,O|\lambda)$
- The Viterbi algorithm (dynamic programming) defines $\delta_j(t)$, i.e., the highest probability of a single path of length t which accounts for the observations and ends in state S_j

$$\delta_j(t) = \max_{q_1,q_2,\ldots,q_{t-1}} P(q_1 q_2 \ldots q_t = j, O_1 O_2 \ldots O_t|\lambda)$$

- By induction

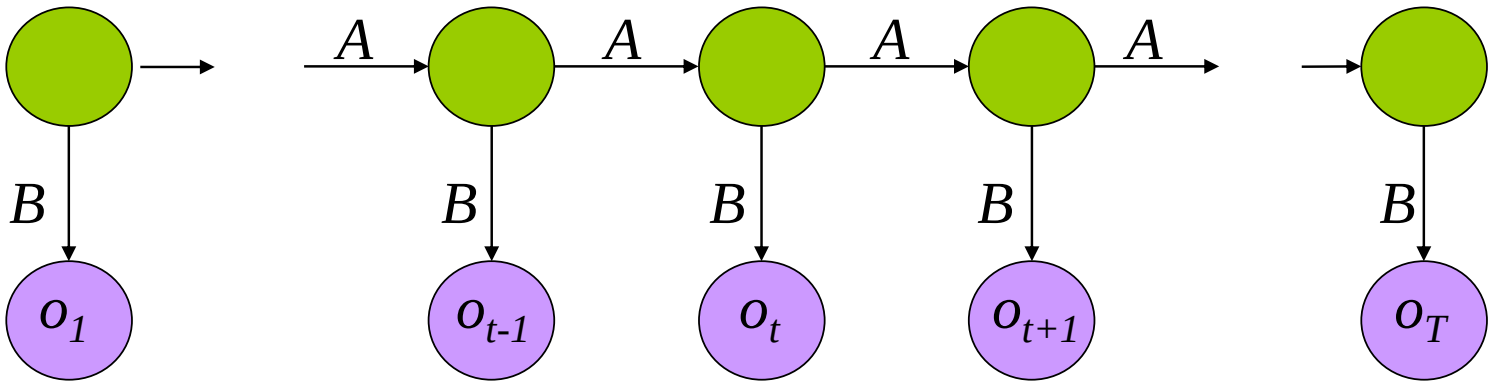
$$\begin{aligned} \delta_j(1) &= \pi_j b_{jO_1} & 1 \le j \le N \\ \delta_j(t+1) &= \left(\max_i \delta_i(t) a_{ij}\right) b_{jO_{t+1}} & 1 \le t \le T-1 \end{aligned}$$

- With backtracking (keeping the maximizing argument for each t and j) we find the optimal solution

Anantharaman Narayana Iyer, Natural Language Processing, Unit 2 – Tagging Problems and HMM (Slides)

Content

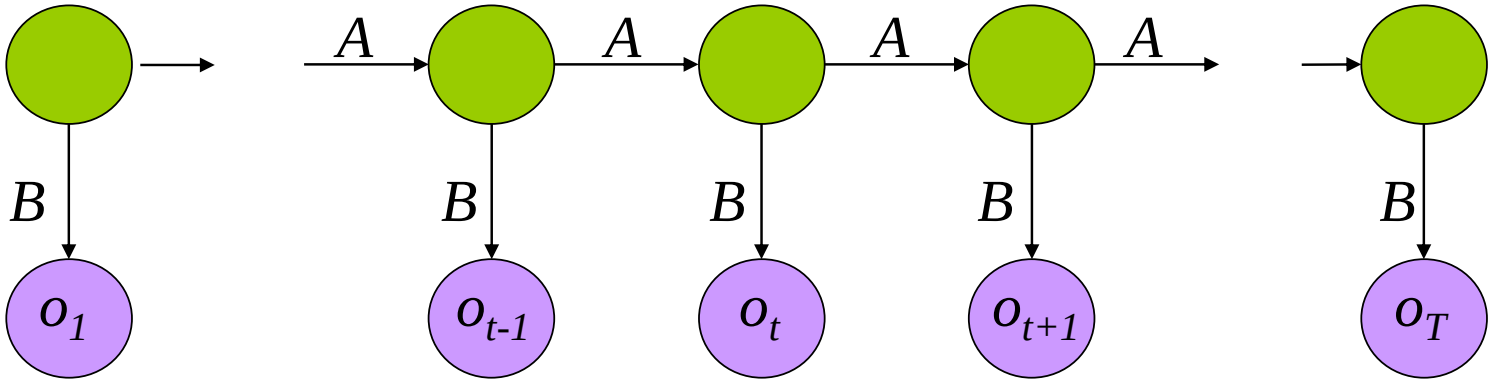
Parameter Estimation



- Given an observation sequence, find the model that is most likely to produce that sequence.
- No analytic method
- Given a model and observation sequence, update the model parameters to better fit the observations.

David Meir Blei, Hidden Markov Models, 1999 (Slides)

Parameter Estimation



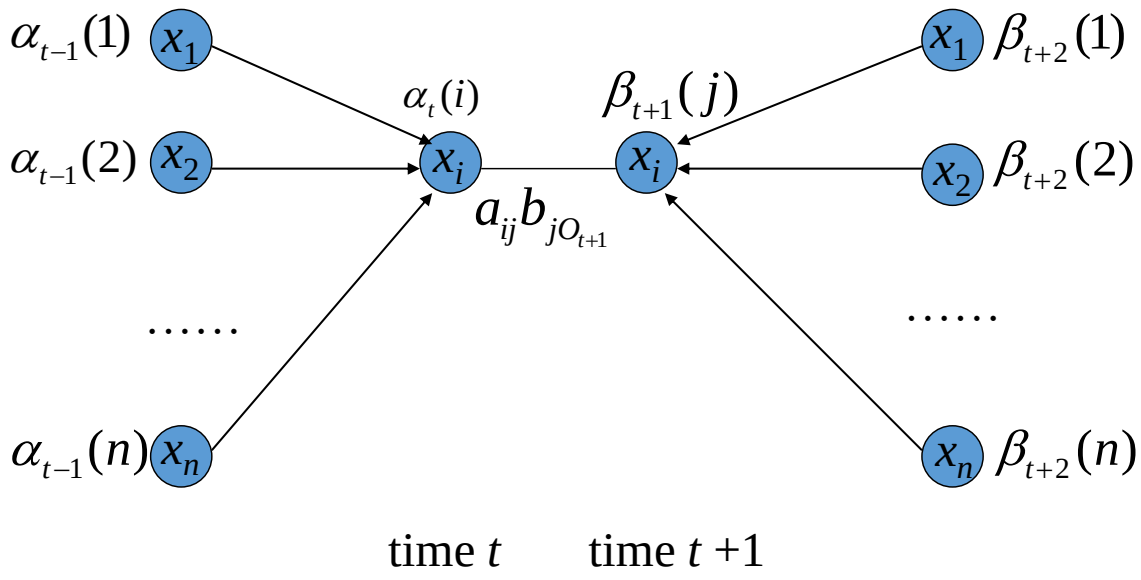
$$p_t(i,j) = \frac{\alpha_i(t) a_{ij} b_{jo_{t+1}} \beta_j(t+1)}{\sum_{m=1\ldots N} \alpha_m(t)\beta_m(t)}$$

Probability of traversing an arc

$$\gamma_i(t) = \sum_{j=1\ldots N} p_t(i,j)$$

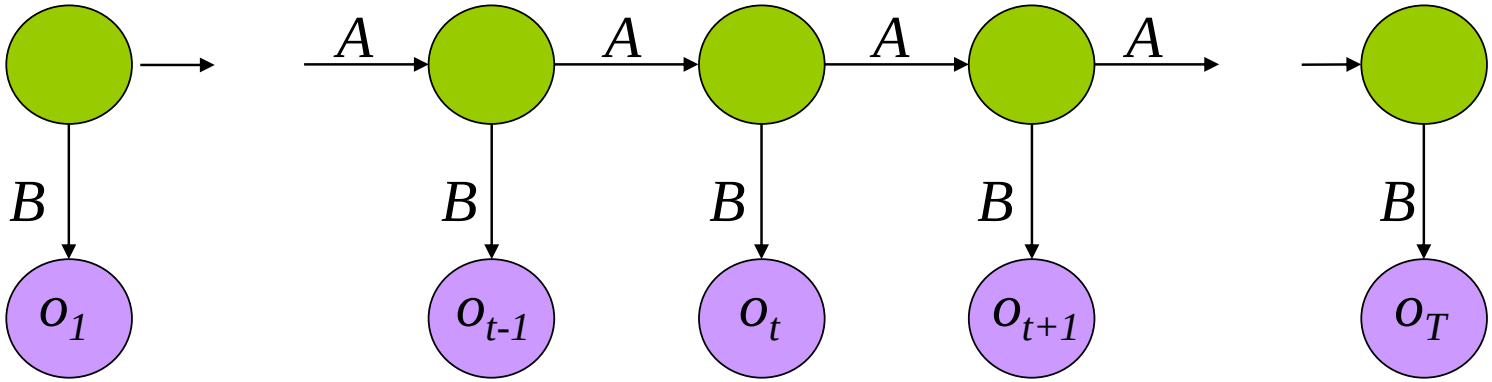
Probability of being in state i

David Meir Blei, Hidden Markov Models, 1999 (Slides)



forward variable and backward variable are used in $\gamma_t(i,j)$

Parameter Estimation



$$\hat{\pi}_i = \gamma_i(1)$$

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

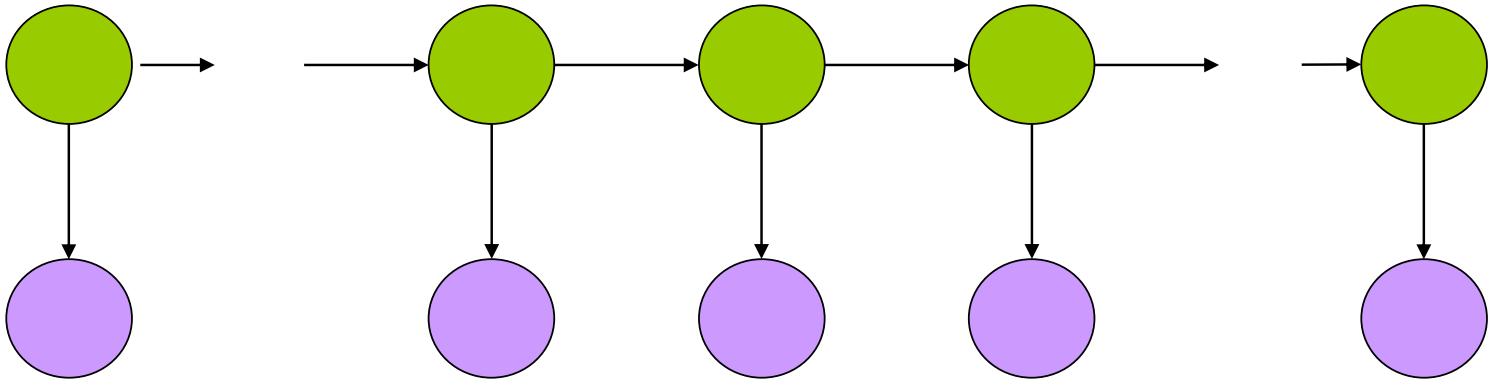
$$\hat{b}_{ik} = \frac{\sum_{\{t:o_t=k\}} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_i(t)}$$

Now we can compute the new estimates of the model parameters.

David Meir Blei, Hidden Markov Models, 1999 (Slides)

Content

HMM Applications



- Generating parameters for n-gram models
- Tagging speech
- Speech recognition

David Meir Blei, Hidden Markov Models, 1999 (Slides)

Content

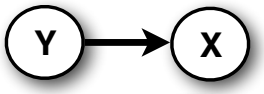
Content

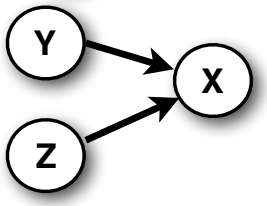
Directed graphical models

Graphical models are a **notation for probability models**.
In a **directed** graphical model, **each node** represents a distribution over a random variable:

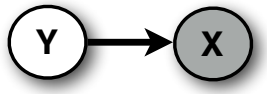
– $P(X) =$ 

Arrows represent dependencies (they define what other random variables the current node is conditioned on)

– $P(Y)\ P(X \mid Y) =$ 

– $P(Y)\ P(Z)\ P(X \mid Y, Z) =$ 

Shaded nodes represent observed variables.
White nodes represent hidden variables

– $P(Y)\ P(X \mid Y)$ with Y hidden and X observed = 

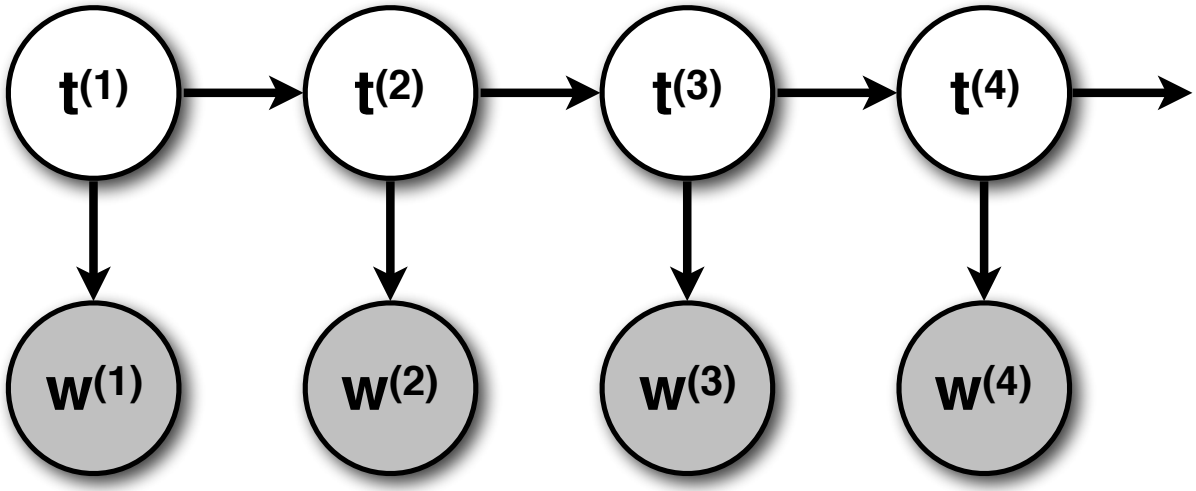
Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

HMMs as graphical models

HMMs are **generative** models of the observed input string **w**

They 'generate' **w** with $P(\mathbf{w},\mathbf{t}) = \prod_i P(t^{(i)}|\, t^{(i-1)}) P(w^{(i)}|\, t^{(i)})$

When we use an HMM to tag, we observe **w**, and need to find **t**



Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Models for sequence labeling

Sequence labeling: Given an input sequence $\mathbf{w} = w^{(1)} \ldots w^{(n)}$, predict the best (most likely) label sequence $\mathbf{t} = t^{(1)} \ldots t^{(n)}$

$$\underset{\mathbf{t}}{\operatorname{argmax}} P(\mathbf{t}|\mathbf{w})$$

Generative models use Bayes Rule:

$$\begin{aligned}
\underset{\mathbf{t}}{\operatorname{argmax}} P(\mathbf{t}|\mathbf{w}) &= \underset{\mathbf{t}}{\operatorname{argmax}} \frac{P(\mathbf{t},\mathbf{w})}{P(\mathbf{w})} \\
&= \underset{\mathbf{t}}{\operatorname{argmax}} P(\mathbf{t},\mathbf{w}) \\
&= \underset{\mathbf{t}}{\operatorname{argmax}} P(\mathbf{t})P(\mathbf{w}|\mathbf{t})
\end{aligned}$$

Discriminative (conditional) models model $\mathrm{P}(\mathbf{t}\,|\mathbf{w})$ directly

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Advantages of discriminative models

We're usually not really interested in $\mathrm{P}(\mathbf{w} \mid \mathbf{t})$.

– $\mathbf{w}$ is given. We don't need to predict it!

Why not model what we're actually interested in: $\mathrm{P}(\mathbf{t} \mid \mathbf{w})$

Modeling $\mathrm{P}(\mathbf{w} \mid \mathbf{t})$ well is quite difficult:

– Prefixes (capital letters) or suffixes are good predictors for certain classes of $\mathbf{t}$ (proper nouns, adverbs,…)

– Se we don't want to model words as atomic symbols, but in terms of features

– These features may also help us deal with unknown words

– But features may not be independent

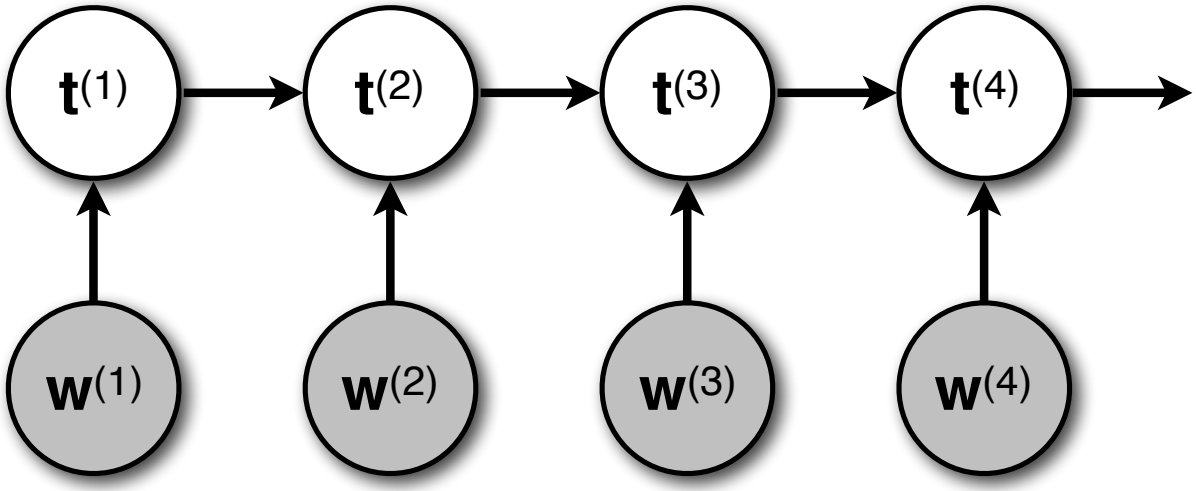
Modeling $\mathrm{P}(\mathbf{t} \mid \mathbf{w})$ with features should be easier:

– Now we can incorporate arbitrary features of the word, because we don't need to predict $\mathbf{w}$ anymore

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Discriminative probability models

A discriminative or **conditional** model of the labels **t** given the observed input string **w** models $P(\mathbf{t} \mid \mathbf{w}) = \prod_i P(t^{(i)} \mid w^{(i)}, t^{(i-1)})$ directly.



Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Discriminative models

There are two main types of discriminative probability models:

- Maximum Entropy Markov Models (MEMMs)
- Conditional Random Fields (CRFs)

MEMMs and CRFs:

- are both based on logistic regression
- have the same graphical model
- require the Viterbi algorithm for tagging
- differ in that MEMMs consist of independently learned distributions, while CRFs are trained to maximize the probability of the entire sequence

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Probabilistic classification

Classification:

Predict a class (label) c for an input $\mathbf{x}$

There are only a (small) finite number of possible class labels

Probabilistic classification:

– Model the probability $P(c \mid \mathbf{x})$

$P(c|\mathbf{x})$ is a probability if $0 \leq P(c_i \mid \mathbf{x}) \leq 1$, and $\sum_i P(c_i \mid \mathbf{x}) = 1$

– Return the class $c^* = \text{argmax}_i P(c_i \mid \mathbf{x})$ that has the highest probability

One standard way to model $P(c \mid \mathbf{x})$ is logistic regression (used by MEMMs and CRFs)

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Using features

Think of feature functions as useful questions you can ask about the input $\mathbf{x}$:

- **Binary feature functions**:
  $f_{\text{first-letter-capitalized}}(\textbf{Urbana}) = 1$
  $f_{\text{first-letter-capitalized}}(\textbf{computer}) = 0$

- **Integer (or real-valued) features**:
  $f_{\text{number-of-vowels}}(\textbf{Urbana}) = 3$

Which specific feature functions are useful will depend on your task (and your training data).

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

From features to probabilities

We associate a real-valued weight w_{ic} with each feature function $f_i(\mathbf{x})$ and output class c

Note that the feature function $f_i(\mathbf{x})$ does not have to depend on c as long as the weight does (note the double index w_{ic})

This gives us a real-valued score for predicting class c for input $\mathbf{x}$: $\text{score}(\mathbf{x},c) = \sum_i w_{ic} f_i(\mathbf{x})$

This score could be negative, so we exponentiate it:
$\text{score}(\mathbf{x},c) = \exp(\sum_i w_{ic} f_i(\mathbf{x}))$
To get a probability distribution over all classes c, we renormalize these scores:

$$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$$
$$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$$

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Learning: finding $\boldsymbol{w}$

Learning = finding weights $\boldsymbol{w}$
We use conditional maximum likelihood estimation
(and standard convex optimization algorithms)
to find/learn $\boldsymbol{w}$

(for more details, attend CS446 and CS546)

The conditional MLE training objective:

Find the **w** that assigns highest probability to all observed outputs c_i given the inputs $\mathbf{x}_i$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_i P(c_i|\mathbf{x}_i, \mathbf{w})$$

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Terminology

Models that are of the form

$$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$$
$$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$$

are also called loglinear models, Maximum Entropy (MaxEnt) models, or multinomial logistic regression models

CS446 and CS546 should give you more details about these.

The normalizing term $\sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$ is also called the partition function and is often abbreviated as Z

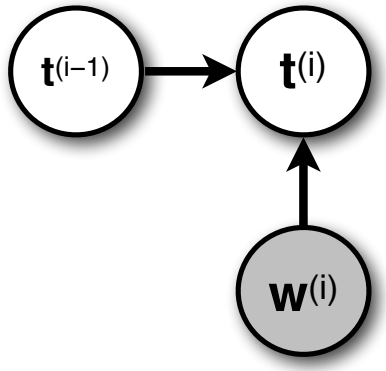
Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Terminology

- All these terms are the same:
  - Softmax
  - Softmax regression
  - Multinomial (or multiclass) logistic regression
  - Maximum Entropy (MaxEnt) model (classifier)
  - Multinomial logit
  - Monditional maximum entropy model

Maximum Entropy Markov Models

MEMMs use a MaxEnt classifier for each $P(t^{(i)} | w^{(i)}, t^{(i-1)})$:



Since we use w to refer to words, let's use λ_{jk} as the weight for the feature function $f_j(t^{(i-1)}, w^{(i)})$ when predicting tag t_k:

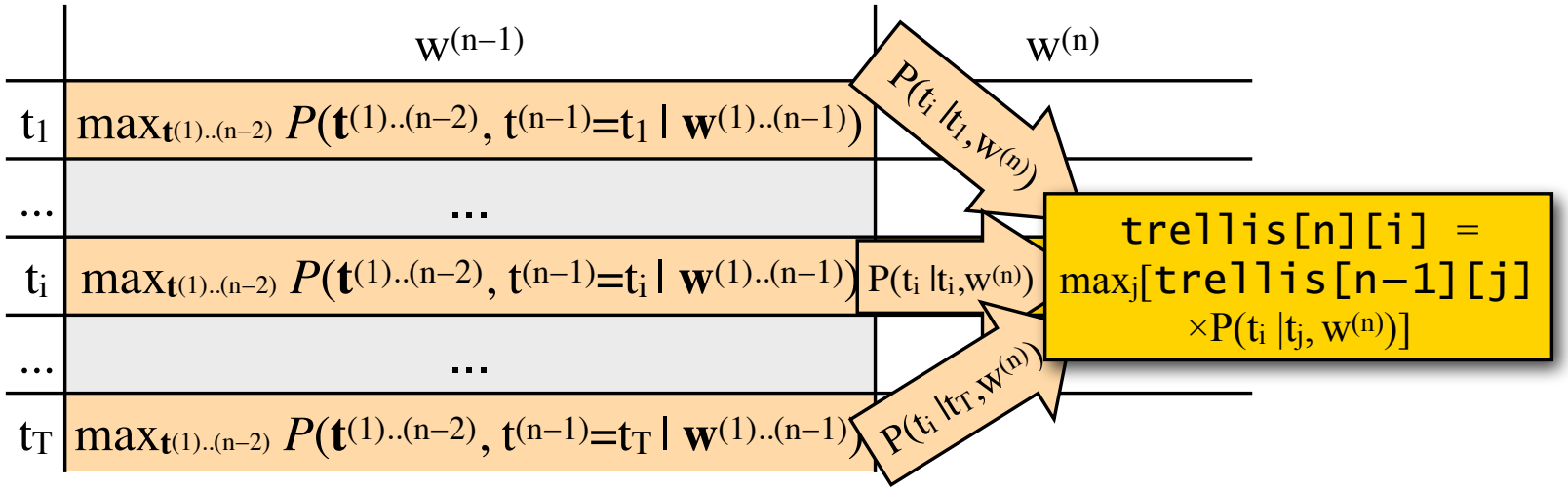
$$P(t^{(i)} = t_k \mid t^{(i-1)}, w^{(i)}) = \frac{\exp(\sum_j \lambda_{jk} f_j(t^{(i-1)}, w^{(i)})}{\sum_l \exp(\sum_j \lambda_{jl} f_j(t^{(i-1)}, w^{(i)})}$$

Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

Viterbi for MEMMs

`trellis[n][i]` stores the probability of the most likely (Viterbi) tag sequence $\mathbf{t}^{(1)\ldots(n)}$ that ends in tag t_i for the prefix $w^{(1)}\ldots w^{(n)}$
Remember that we do not generate $\mathbf{w}$ in MEMMs. So:

$$\texttt{trellis[n][i]} = \max_{\mathbf{t}^{(1)..(n-1)}}[\, P(\mathbf{t}^{(1)\ldots(n-1)}, t^{(n)}{=}t_i \mid \mathbf{w}^{(1)\ldots(n)})\,]$$
$$= \max_j\,[\,\texttt{trellis[n-1][j]} \times P(t_i \mid t_j, w^{(n)})\,]$$
$$= \max_j\,[\max_{\mathbf{t}^{(1)..(n-2)}}[P(\mathbf{t}^{(1)..(n-2)}, t^{(n-1)}{=}t_j \mid \mathbf{w}^{(1)..(n-1)})] \times P(t_i \mid t_j, w^{(n)})]$$



Julia Hockenmaier, UIUC CS447: Natural Language Processing (slides)

The Maximum Entropy Markov Model (MEMM)

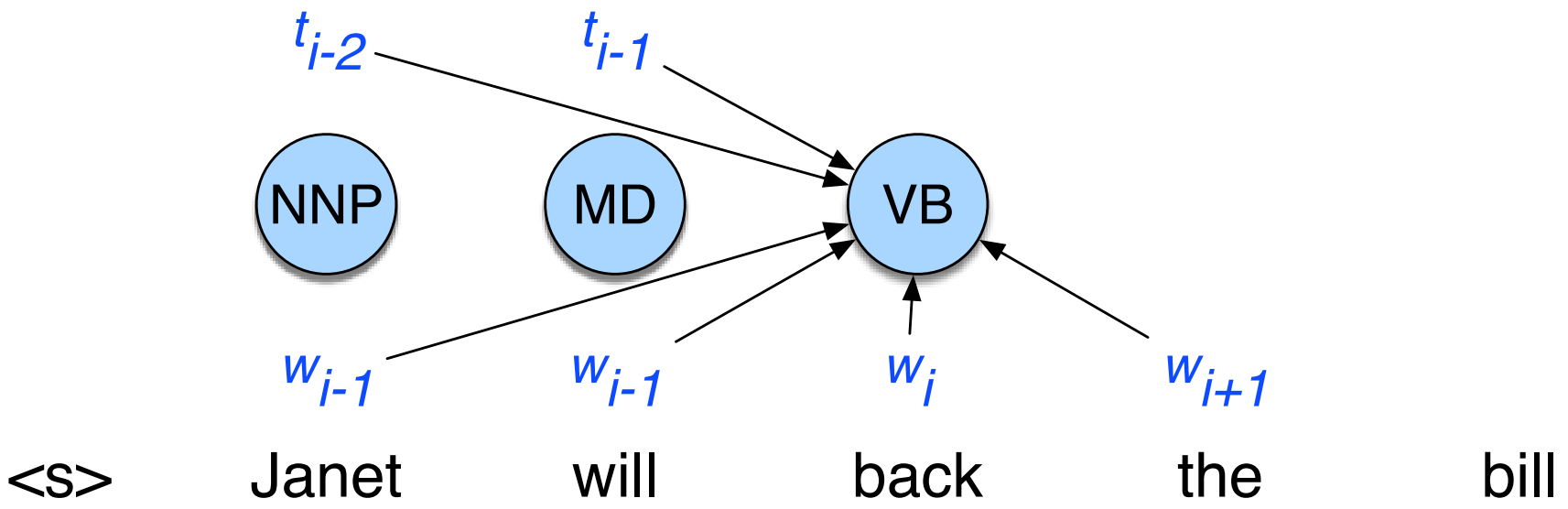
- A sequence version of the logistic regression (also called maximum entropy) classifier.
- Find the best series of tags:

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}$$

48

Daniel Jurafsky and James H. Martin, Part-of-speech tagging (slides)

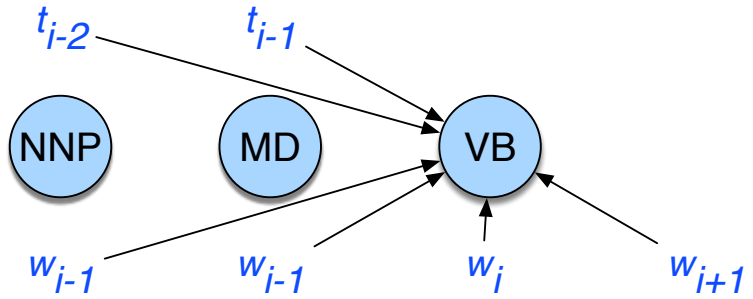
The Maximum Entropy Markov Model (MEMM)



49

Daniel Jurafsky and James H. Martin, Part-of-speech tagging (slides)

Features for the classifier at each tag



<s> Janet will back the bill

$$t_i = \text{VB and } w_{i-2} = \text{Janet}$$
$$t_i = \text{VB and } w_{i-1} = \text{will}$$
$$t_i = \text{VB and } w_i = \text{back}$$
$$t_i = \text{VB and } w_{i+1} = \text{the}$$
$$t_i = \text{VB and } w_{i+2} = \text{bill}$$
$$t_i = \text{VB and } t_{i-1} = \text{MD}$$
$$t_i = \text{VB and } t_{i-1} = \text{MD and } t_{i-2} = \text{NNP}$$
$$t_i = \text{VB and } w_i = \text{back and } w_{i+1} = \text{the}$$

50

Daniel Jurafsky and James H. Martin, Part-of-speech tagging (slides)

More features

w_i contains a particular prefix (from all prefixes of length ≤ 4)
w_i contains a particular suffix (from all suffixes of length ≤ 4)
w_i contains a number
w_i contains an upper-case letter
w_i contains a hyphen
w_i is all upper case
w_i's word shape
w_i's short word shape
w_i is upper case and has a digit and a dash (like *CFC-12*)
w_i is upper case and followed within 3 words by Co., Inc., etc.

51

Daniel Jurafsky and James H. Martin, Part-of-speech tagging (slides)

MEMM computes the best tag sequence

$$
\begin{aligned}
\hat{T} &= \operatorname*{argmax}_T P(T|W) \\
&= \operatorname*{argmax}_T \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\
&= \operatorname*{argmax}_T \prod_i \frac{\exp\left(\sum_i w_i f_i(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}{\sum_{t' \in \text{tagset}} \exp\left(\sum_i w_i f_i(t', w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}
\end{aligned}
$$

52

Daniel Jurafsky and James H. Martin, Part-of-speech tagging (slides)

MEMM Decoding

- Simplest algorithm:

function GREEDY MEMM DECODING(words W, model P) **returns** tag sequence T

for $i = 1$ **to** *length(W)*

$$\hat{t}_i = \underset{t' \in T}{\text{argmax}}\ P(t' \mid w_{i-l}^{i+l}, t_{i-k}^{i-1})$$

- What we use in practice: The **Viterbi** algorithm

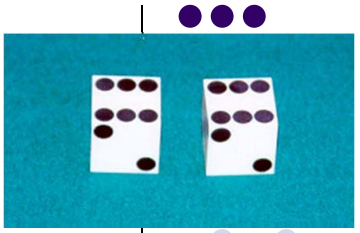
ısed

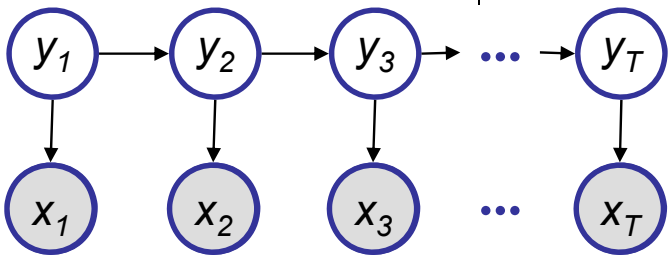
53

Daniel Jurafsky and James H. Martin, Part-of-speech tagging (slides)

Content

Hidden Markov Model revisit





- Transition probabilities between any two states

$$p(y_t^j = 1 \mid y_{t-1}^i = 1) = a_{i,j},$$

or $$p(y_t \mid y_{t-1}^i = 1) \sim \text{Multinomial}(a_{i,1}, a_{i,2}, \ldots, a_{i,M}), \forall i \in \mathbb{I}.$$

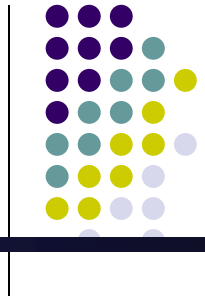
- Start probabilities

$$p(y_1) \sim \text{Multinomial}(\pi_1, \pi_2, \ldots, \pi_M).$$

- Emission probabilities associated with each state

$$p(x_t \mid y_t^i = 1) \sim \text{Multinomial}(b_{i,1}, b_{i,2}, \ldots, b_{i,K}), \forall i \in \mathbb{I}.$$

or in general: $$p(x_t \mid y_t^i = 1) \sim \text{f}(\cdot \mid \theta_i), \forall i \in \mathbb{I}.$$



Inference (review)

- Forward algorithm

$$\alpha_t^k \overset{\text{def}}{=} \mu_{t-1\to t}(k) = P(x_1,\ldots,x_{t-1},x_t,y_t^k=1)$$

$$\alpha_t^k = p(x_t \mid y_t^k=1)\sum_i \alpha_{t-1}^i a_{i,k}$$

- Backward algorithm

$$\beta_t^k = \sum_i a_{k,i}\, p(x_{t+1} \mid y_{t+1}^i = 1)\beta_{t+1}^i$$

$$\beta_t^k \overset{\text{def}}{=} \mu_{t-1\leftarrow t}(k) = P(x_{t+1},\ldots,x_T \mid y_t^k=1)$$

$$\gamma_t^i \overset{\text{def}}{=} p(y_t^i=1 \mid x_{1:T}) \propto \alpha_t^i\beta_t^i = \sum_j \xi_t^{i,j}$$

$$\xi_t^{i,j} \overset{\text{def}}{=} p(y_t^i=1, y_{t+1}^j=1, x_{1:T})$$

$$\propto \mu_{t-1\to t}(y_t^i=1)\mu_{t\leftarrow t+1}(y_{t+1}^j=1)\,p(x_{t+1}\mid y_{t+1})\,p(y_{t+1}\mid y_t)$$

$$\xi_t^{i,j} = \alpha_t^i\beta_{t+1}^j a_{i,j}\, p(x_{t+1}\mid y_{t+1}^i=1)$$

The matrix-vector form:

$$B_t(i) \overset{\text{def}}{=} p(x_t \mid y_t^i = 1)$$

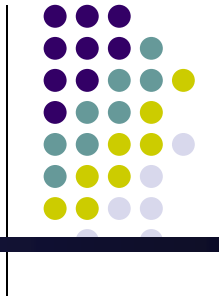
$$A(i,j) \overset{\text{def}}{=} p(y_{t+1}^j = 1 \mid y_t^i = 1)$$

$$\alpha_t = \left(A^T\alpha_{t-1}\right).*B_t$$

$$\beta_t = A\left(\beta_{t+1}.*B_{t+1}\right)$$

$$\xi_t = \left(\alpha_t\left(\beta_{t+1}.*B_{t+1}\right)^T\right).*A$$

$$\gamma_t = \alpha_t .*\beta_t$$



Learning HMM

- **Supervised learning**: estimation when the "right answer" is known
  - **Examples:**

    GIVEN: a genomic region $x = x_1 \quad x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands

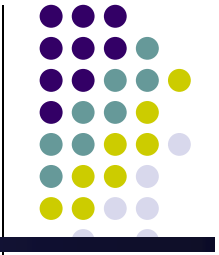
    GIVEN: the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

- **Unsupervised learning**: estimation when the "right answer" is unknown
  - **Examples:**

    GIVEN: the porcupine genome; we don't know how frequent are the CpG islands there, neither do we know their composition

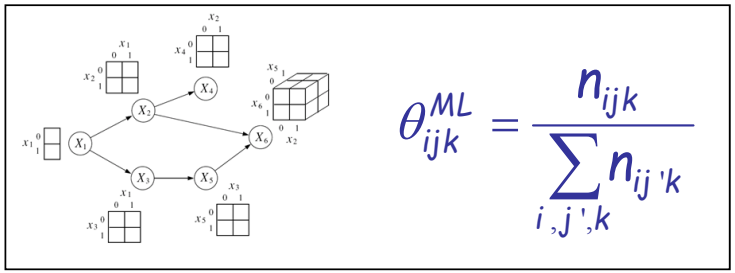
    GIVEN: 10,000 rolls of the casino player, but we don't see when he changes dice

- **QUESTION:** Update the parameters θ of the model to maximize $P(x|\theta)$ -- Maximal likelihood (ML) estimation



Learning HMM: two scenarios

- Supervised learning: if only we knew the true state path then ML parameter estimation would be trivial
  - E.g., recall that for complete observed tabular BN:

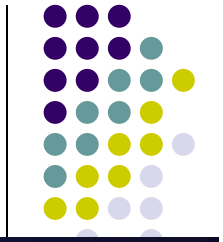


$$\theta_{ijk}^{ML} = \frac{n_{ijk}}{\sum_{i,j',k} n_{ij'k}}$$

$$a_{ij}^{ML} = \frac{\#(i \to j)}{\#(i \to \bullet)} = \frac{\sum_n \sum_{t=2}^{T} y_{n,t-1}^{i} y_{n,t}^{j}}{\sum_n \sum_{t=2}^{T} y_{n,t-1}^{i}}$$

$$b_{ik}^{ML} = \frac{\#(i \to k)}{\#(i \to \bullet)} = \frac{\sum_n \sum_{t=1}^{T} y_{n,t}^{i} x_{n,t}^{k}}{\sum_n \sum_{t=1}^{T} y_{n,t}^{i}}$$

  - What if y is continuous? We can treat $\{(x_{n,t}, y_{n,t}) : t = 1:T, n = 1:N\}$ as $N \times T$ observations of, e.g., a GLIM, and apply learning rules for GLIM
- Unsupervised learning: when the true state path is unknown, we can fill in the missing values using inference recursions.
  - The Baum Welch algorithm (i.e., EM)
    - Guaranteed to increase the log likelihood of the model after each iteration
    - Converges to local optimum, depending on initial conditions



The Baum Welch algorithm

- The complete log likelihood

$$\ell_c(\boldsymbol{\theta};\mathbf{x},\mathbf{y}) = \log p(\mathbf{x},\mathbf{y}) = \log \prod_n \left(p(y_{n,1}) \prod_{t=2}^{T} p(y_{n,t} \mid y_{n,t-1}) \prod_{t=1}^{T} p(x_{n,t} \mid x_{n,t}) \right)$$

- The expected complete log likelihood

$$\left\langle \ell_c(\boldsymbol{\theta};\mathbf{x},\mathbf{y}) \right\rangle = \sum_n \left(\left\langle y_{n,1}^i \right\rangle_{p(y_{n,1}|\mathbf{x}_n)} \log \pi_i \right) + \sum_n \sum_{t=2}^{T} \left(\left\langle y_{n,t-1}^i y_{n,t}^j \right\rangle_{p(y_{n,t-1},y_{n,t}|\mathbf{x}_n)} \log a_{i,j} \right) + \sum_n \sum_{t=1}^{T} \left(x_{n,t}^k \left\langle y_{n,t}^i \right\rangle_{p(y_{n,t}|\mathbf{x}_n)} \log b_{i,k} \right)$$

- EM
  - The E step

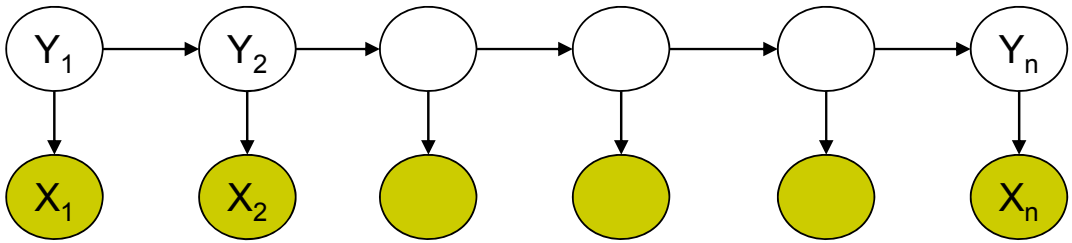
$$\gamma_{n,t}^i = \left\langle y_{n,t}^i \right\rangle = p(y_{n,t}^i = 1 \mid \mathbf{x}_n)$$

$$\xi_{n,t}^{i,j} = \left\langle y_{n,t-1}^i y_{n,t}^j \right\rangle = p(y_{n,t-1}^i = 1, y_{n,t}^j = 1 \mid \mathbf{x}_n)$$

  - The M step ("symbolically" identical to MLE)

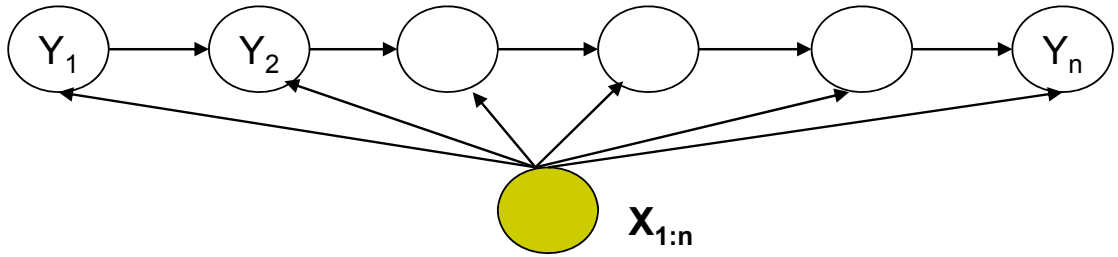
$$\pi_i^{ML} = \frac{\sum_n \gamma_{n,1}^i}{N} \qquad a_{ij}^{ML} = \frac{\sum_n \sum_{t=2}^{T} \xi_{n,t}^{i,j}}{\sum_n \sum_{t=1}^{T-1} \gamma_{n,t}^i} \qquad b_{ik}^{ML} = \frac{\sum_n \sum_{t=1}^{T} \gamma_{n,t}^i x_{n,t}^k}{\sum_n \sum_{t=1}^{T-1} \gamma_{n,t}^i}$$

Shortcomings of Hidden Markov Model (1): locality of features



- HMM models capture dependences between each state and only its corresponding observation
  - NLP example: In a sentence segmentation task, each segmental state may depend not just on a single word (and the adjacent segmental stages), but also on the (non-local) features of the whole line such as line length, indentation, amount of white space, etc.
- Mismatch between learning objective function and prediction objective function
  - HMM learns a joint distribution of states and observations P(**Y**, **X**), but in a prediction task, we need the conditional probability P(**Y**|**X**)

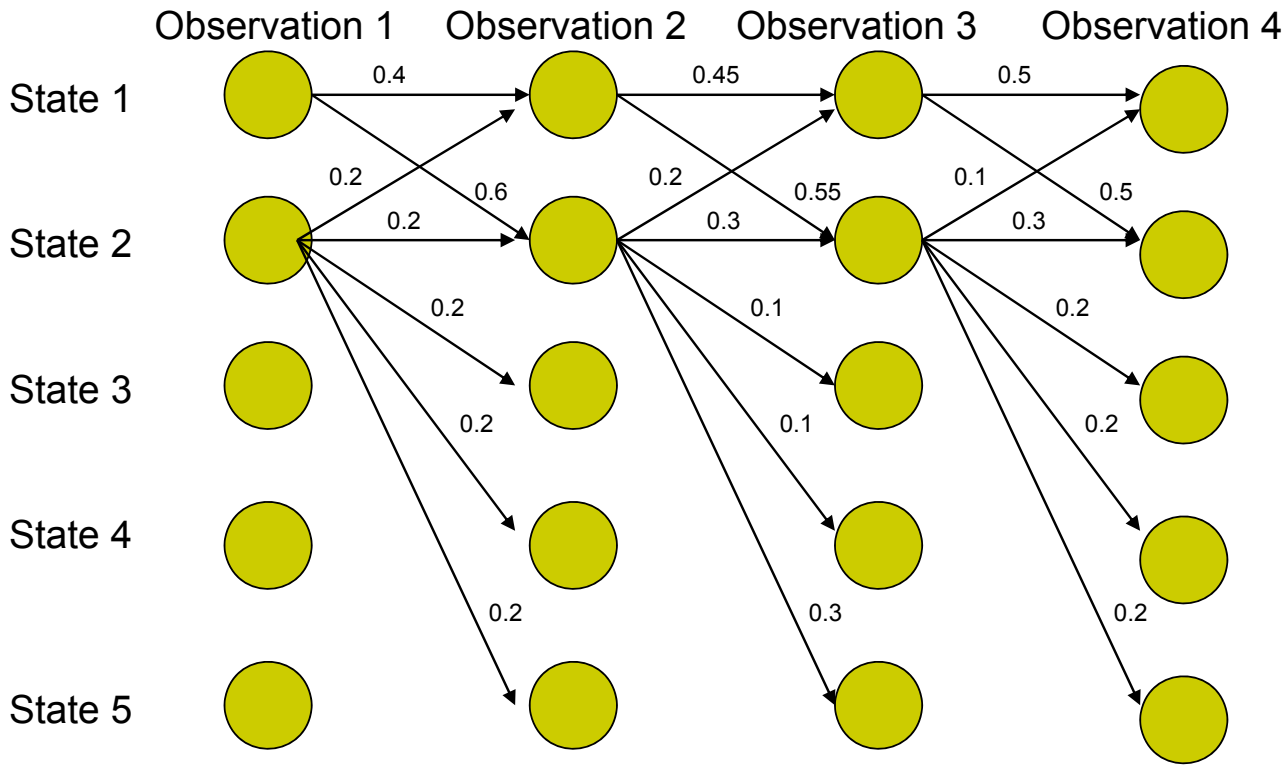
Solution:
Maximum Entropy Markov Model (MEMM)



$$P(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}) = \prod_{i=1}^{n} P(y_i|y_{i-1},\mathbf{x}_{1:n}) = \prod_{i=1}^{n} \frac{\exp(\mathbf{w}^T\mathbf{f}(y_i,y_{i-1},\mathbf{x}_{1:n}))}{Z(y_{i-1},\mathbf{x}_{1:n})}$$

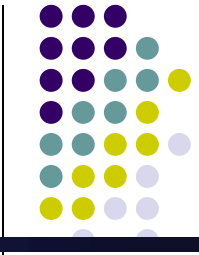
- Models dependence between each state and the full observation sequence explicitly
  - More expressive than HMMs
- Discriminative model
  - Completely ignores modeling P(**X**): saves modeling effort
  - Learning objective function consistent with predictive function: P(**Y**|**X**)

Then, shortcomings of MEMM (and HMM) (2): the Label bias problem

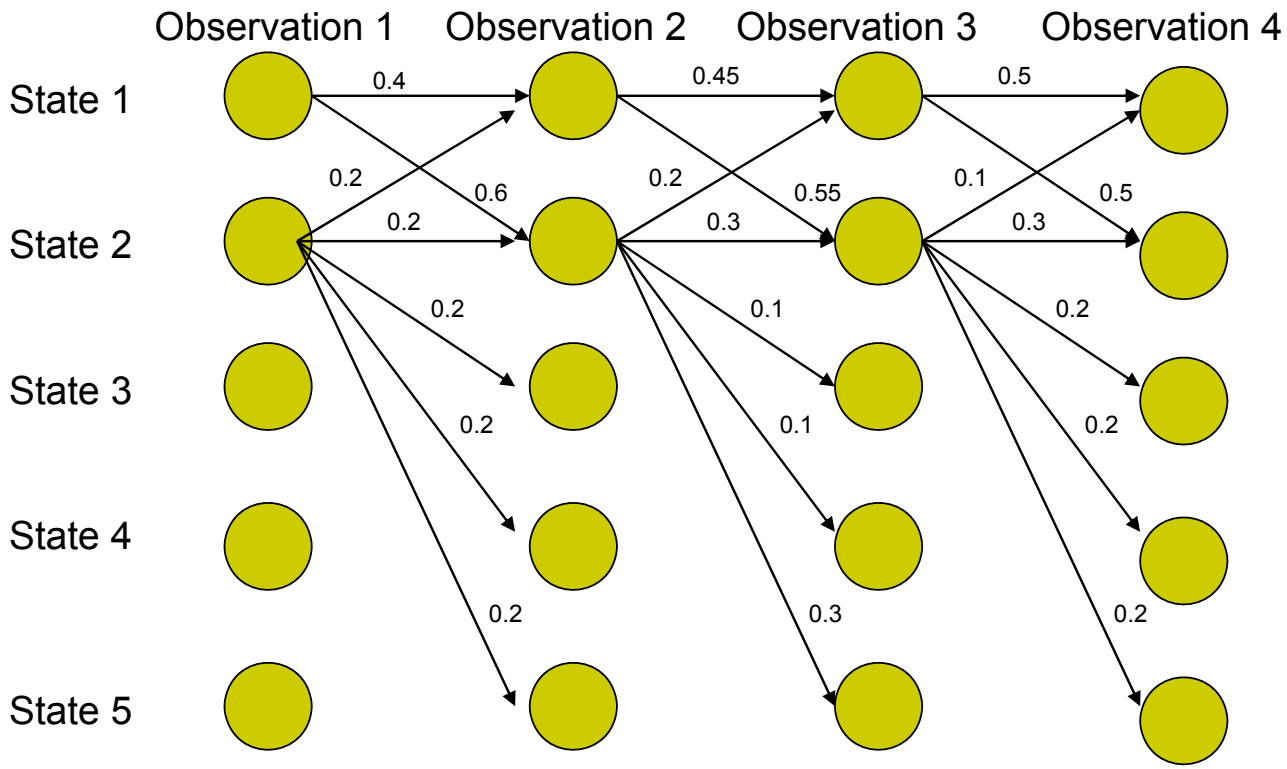


What the local transition probabilities say:

- State 1 almost always prefers to go to state 2
- State 2 almost always prefer to stay in state 2



MEMM: the Label bias problem



Probability of path 1-> 1-> 1-> 1:

- 0.4 x 0.45 x 0.5 = 0.09