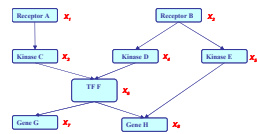


Hidden Markov Model and Conditional Random Fields

Probabilistic Graphical Models (10-708)

Lecture 12, Oct 29, 2007



Eric Xing

Reading: J-Chap. 12, and addition papers

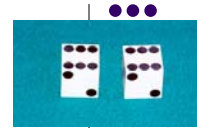
1



- Feedbacks
 - Today
 - Office hour
- Recitation

- Project milestone
 - This Wednesday

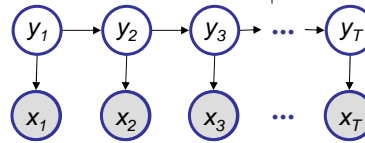
Hidden Markov Model revisit



- Transition probabilities between any two states

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

or $p(y_t | y_{t-1} = \mathbf{1}) \sim \text{Multinomial}(a_{i,1}, a_{i,2}, \dots, a_{i,M}), \forall i \in I.$



- Start probabilities

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

- Emission probabilities associated with each state

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

or in general: $p(x_t | y_t = \mathbf{1}) \sim f(\cdot | \theta_t), \forall t \in I.$

Eric Xing

3

Applications of HMMs



- **Some early applications of HMMs**

- finance, but we never saw them
- speech recognition
- modelling ion channels

- **In the mid-late 1980s HMMs entered genetics and molecular biology, and they are now firmly entrenched.**

- **Some current applications of HMMs to biology**

- mapping chromosomes
- aligning biological sequences
- predicting sequence structure
- inferring evolutionary relationships
- finding genes in DNA sequence

Eric Xing

4

The Forward Algorithm



- We want to calculate $P(\mathbf{x})$, the likelihood of \mathbf{x} , given the HMM
- Sum over all possible ways of generating \mathbf{x} :

$$p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}) = \sum_{y_1} \sum_{y_2} \cdots \sum_{y_N} \pi_{y_1} \prod_{t=2}^T a_{y_{t-1}y_t} \prod_{t=1}^T p(x_t | y_t)$$

- To avoid summing over an exponential number of paths \mathbf{y} , define

$$\alpha(y_t^k = \mathbf{1}) = \alpha_t^k \stackrel{\text{def}}{=} P(x_1, \dots, x_t, y_t^k = \mathbf{1}) \quad (\text{the forward probability})$$

- The recursion:

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

$$P(\mathbf{x}) = \sum_k \alpha_T^k$$

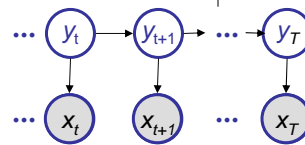
Eric Xing

7

The Backward Algorithm

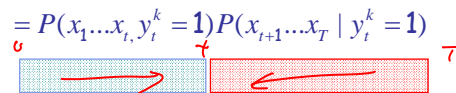


- We want to compute $P(y_t^k = \mathbf{1} | \mathbf{x})$, the posterior probability distribution on the t^{th} position, given \mathbf{x}



- We start by computing

$$P(y_t^k = \mathbf{1} | \mathbf{x}) = P(x_1, \dots, x_t, y_t^k = \mathbf{1}, x_{t+1}, \dots, x_T) \\ \stackrel{P(\mathbf{x})}{=} P(x_1, \dots, x_t, y_t^k = \mathbf{1}) P(x_{t+1}, \dots, x_T | x_1, \dots, x_t, y_t^k = \mathbf{1})$$



Forward, α_t^k Backward, $\beta_t^k = P(x_{t+1}, \dots, x_T | y_t^k = \mathbf{1})$

- The recursion: $\beta_t^k = \sum_i a_{k,i} p(x_{t+1} | y_{t+1}^i = \mathbf{1}) \beta_{t+1}^i$

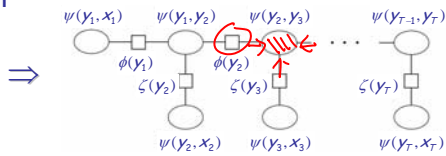
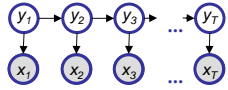
Eric Xing

8

The Junction tree algorithm

$$\phi(x_i) = \phi^i(y_i) \prod_{M \ni x_i} \mu_{\rightarrow x_i}$$

- A junction tree for the HMM



- Rightward pass

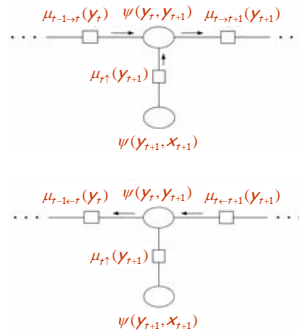
$$\begin{aligned} \mu_{t \rightarrow t+1}(y_{t+1}) &= \sum_{y_t} \psi(y_t, y_{t+1}) \mu_{t-1 \rightarrow t}(y_t) \mu_{t \uparrow}(y_{t+1}) \\ &= \sum_{y_t} p(y_{t+1} | y_t) \mu_{t-1 \rightarrow t}(y_t) p(x_{t+1} | y_{t+1}) \\ &= p(x_{t+1} | y_{t+1}) \sum_{y_t} a_{y_t, y_{t+1}} \mu_{t-1 \rightarrow t}(y_t) \end{aligned}$$

This is exactly the forward algorithm!

- Leftward pass ...

$$\begin{aligned} \mu_{t-1 \leftarrow t}(y_t) &= \sum_{y_{t+1}} \psi(y_t, y_{t+1}) \mu_{t \leftarrow t+1}(y_{t+1}) \mu_{t \uparrow}(y_t) \\ &= \sum_{y_{t+1}} p(y_{t+1} | y_t) \mu_{t \leftarrow t+1}(y_{t+1}) p(x_{t+1} | y_{t+1}) \end{aligned}$$

This is exactly the backward algorithm!



Eric Xing

9

Summary of the F-B algorithm

$$\alpha_t^k \stackrel{\text{def}}{=} \mu_{t-1 \rightarrow t}(k) = P(x_1, \dots, x_{t-1}, x_t, y_t^k = 1)$$

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

$$\beta_t^k \stackrel{\text{def}}{=} \mu_{t-1 \leftarrow t}(k) = P(x_{t+1}, \dots, x_T | y_t^k = 1)$$

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

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

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

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

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

The matrix-vector form:

$$\alpha_t = A_t \alpha_{t-1} \cdot B_t$$

$$\beta_t = A_t \beta_{t+1} \cdot B_{t+1}$$

$$\gamma_t = \alpha_t \cdot \beta_t$$

Handwritten notes: $p(y_t) \alpha_t$, $p(y_t, y_{t+1})$, $\alpha_t = (A^T \alpha_{t-1}) \cdot B_t$, $\beta_t = A(\beta_{t+1} \cdot B_{t+1})$, $\gamma_t = \alpha_t \cdot \beta_t$

Eric Xing

10

Posterior decoding



- We can now calculate

$$P(y_t^k = 1 | \mathbf{x}) = \frac{P(y_t^k = 1, \mathbf{x})}{P(\mathbf{x})} = \frac{\alpha_t^k \beta_t^k}{P(\mathbf{x})}$$

- Then, we can ask

- What is the most likely state at position t of sequence \mathbf{x} :

$$k_t^* = \arg \max_k P(y_t^k = 1 | \mathbf{x})$$

- Note that this is an MPA of a **single** hidden state, what if we want a MPA of a whole hidden state sequence?

- Posterior Decoding: $\{y_t^{k_t^*} = 1 : t = 1 \dots T\}$

- This is different from MPA of a **whole sequence** of hidden states

- This can be understood as **bit error rate** vs. **word error rate**

Example:
MPA of X ?
MPA of (X, Y) ?

x	y	$P(x, y)$
0	0	0.35
0	1	0.05
1	0	0.3
1	1	0.3

Viterbi decoding

- GIVEN $\mathbf{x} = x_1, \dots, x_T$, we want to find $\mathbf{y} = y_1, \dots, y_T$, such that $P(\mathbf{y} | \mathbf{x})$ is maximized:

$$y^* = \arg \max_{\mathbf{y}} P(\mathbf{y} | \mathbf{x}) = \arg \max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x})$$

- Let

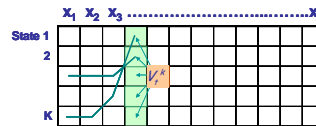
$$V_t^k = \max_{\{y_1, \dots, y_{t-1}\}} P(x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}, x_t, y_t^k = 1)$$

= Probability of most likely **sequence of states** ending at state $y_t = k$

- The recursion:

$$V_t^k = p(x_t | y_t^k = 1) \max_i a_{i,k} V_{t-1}^i$$

- Underflows are a significant problem



$$P(x_1, \dots, x_t, y_1, \dots, y_t) = \pi_{y_1} a_{y_1, y_2} \dots a_{y_{t-1}, y_t} b_{y_t, x_t}$$

- These numbers become extremely small – **underflow**
- Solution: Take the logs of all values: $V_t^k = \log p(x_t | y_t^k = 1) + \max_i (\log(a_{i,k}) + V_{t-1}^i)$

The Viterbi Algorithm – derivation



- Define the viterbi probability:

$$\begin{aligned}
 V_{t+1}^k &= \max_{\{y_1, \dots, y_t\}} \mathcal{P}(x_1, \dots, x_t, y_1, \dots, y_t, x_{t+1}, y_{t+1}^k = 1) \\
 &= \max_{\{y_1, \dots, y_t\}} \mathcal{P}(x_{t+1}, y_{t+1}^k = 1 | x_1, \dots, x_t, y_1, \dots, y_t) \mathcal{P}(x_1, \dots, x_t, y_1, \dots, y_t) \\
 &= \max_{\{y_1, \dots, y_t\}} \mathcal{P}(x_{t+1}, y_{t+1}^k = 1 | y_t) \mathcal{P}(x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}, x_t, y_t) \\
 &= \max_i \mathcal{P}(x_{t+1}, y_{t+1}^k = 1 | y_t^i = 1) \max_{\{y_1, \dots, y_{t-1}\}} \mathcal{P}(x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}, x_t, y_t^i = 1) \\
 &= \max_i \mathcal{P}(x_{t+1}, | y_{t+1}^k = 1) a_{i,k} V_t^i \\
 &= \mathcal{P}(x_{t+1}, | y_{t+1}^k = 1) \max_i a_{i,k} V_t^i
 \end{aligned}$$

Eric Xing

13

Computational Complexity and implementation details



- What is the running time, and space required, for Forward, and Backward?

$$\begin{aligned}
 \alpha_t^k &= \mathcal{P}(x_t | y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k} \\
 \beta_t^k &= \sum_i a_{k,i} \mathcal{P}(x_{t+1} | y_{t+1}^i = 1) \beta_{t+1}^i \\
 V_t^k &= \mathcal{P}(x_t | y_t^k = 1) \max_i a_{i,k} V_{t-1}^i
 \end{aligned}$$

Time: $O(k^2 N)$; Space: $O(kN)$.

- Useful implementation technique to avoid underflows
 - Viterbi: sum of logs
 - Forward/Backward: rescaling at each position by multiplying by a constant

Eric Xing

14

Learning HMM

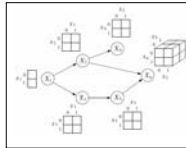


- **Supervised learning:** estimation when the “right answer” is known
 - **Examples:**
 - GIVEN:** a genomic region $x = x_1 \dots x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands
 - GIVEN:** the casino player allows us to observe him one evening, as he changes dice and 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:



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

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

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

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

- The expected complete log likelihood

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

- EM

- The E step

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

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

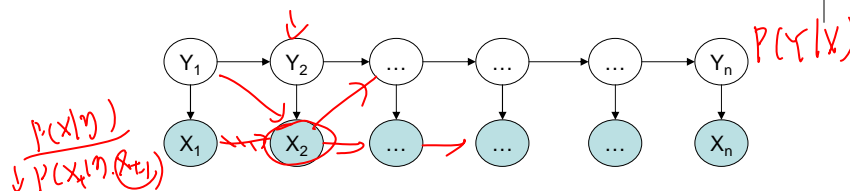
- The M step ("symbolically" identical to MLE)

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

Eric Xing

17

Shortcomings of Hidden Markov Model



- HMM models capture dependencies 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

$$f(\mathbf{x}, \mathbf{y}) \rightarrow p(\mathbf{x}) \rightarrow p(\mathbf{y} | \mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})}$$

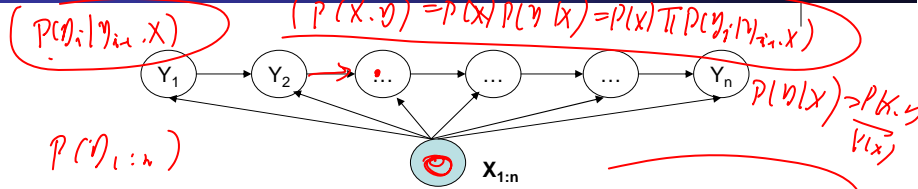
- HMM learns a joint distribution of states and observations $P(\mathbf{Y}, \mathbf{X})$, but in a prediction task, we need the conditional probability $P(\mathbf{Y} | \mathbf{X})$

Eric Xing

18

Solution:

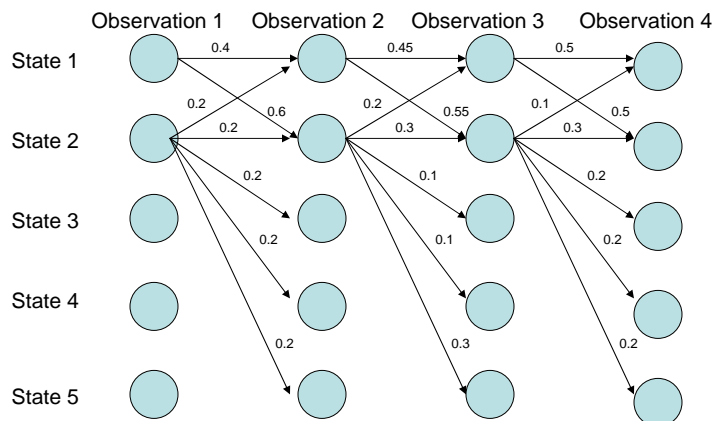
Maximum Entropy Markov Model (MEMM)



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

- 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: $P(Y|X)$

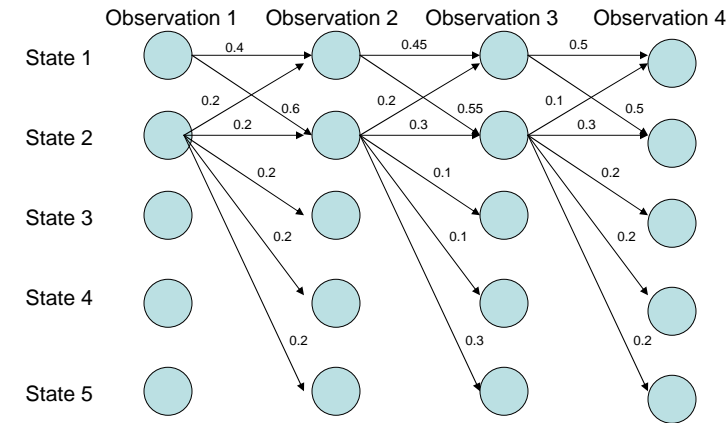
MEMM: the Label bias problem



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



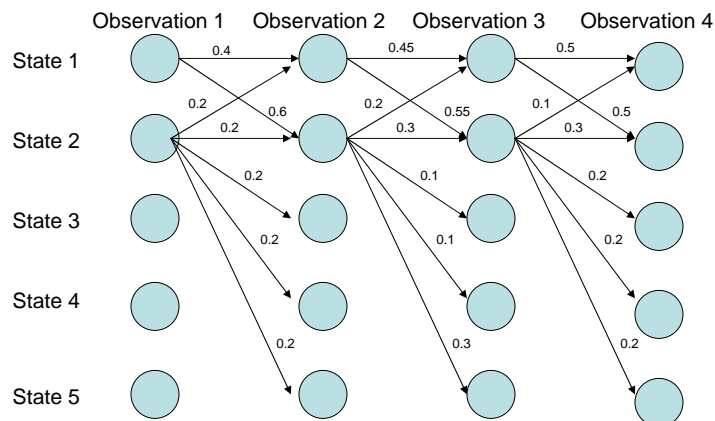
Probability of path 1-> 1-> 1-> 1:

- $0.4 \times 0.45 \times 0.5 = 0.09$

Eric Xing

21

MEMM: the Label bias problem



Probability of path 2->2->2->2 :

- $0.2 \times 0.3 \times 0.3 = 0.018$

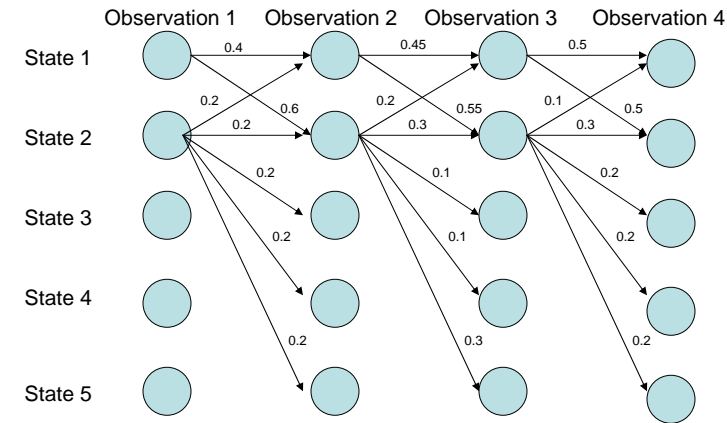
Other paths:

- 1-> 1-> 1-> 1: 0.09

Eric Xing

22

MEMM: the Label bias problem



Probability of path 1->2->1->2:

- $0.6 \times 0.2 \times 0.5 = 0.06$

Other paths:

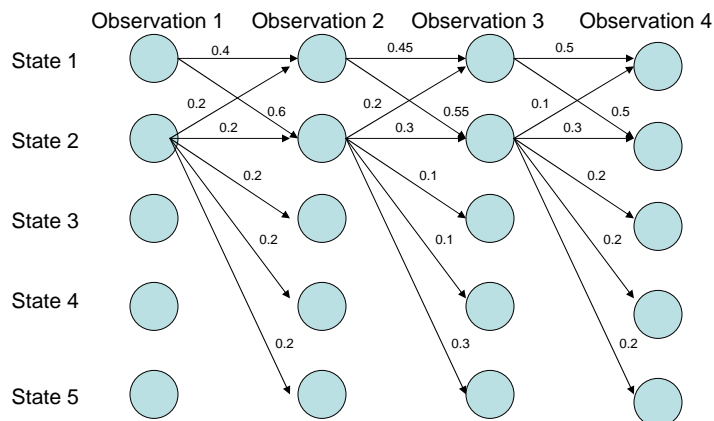
1->1->1->1: 0.09

2->2->2->2: 0.018

Eric Xing

23

MEMM: the Label bias problem



Probability of path 1->1->2->2:

- $0.4 \times 0.55 \times 0.3 = 0.066$

Other paths:

1->1->1->1: 0.09

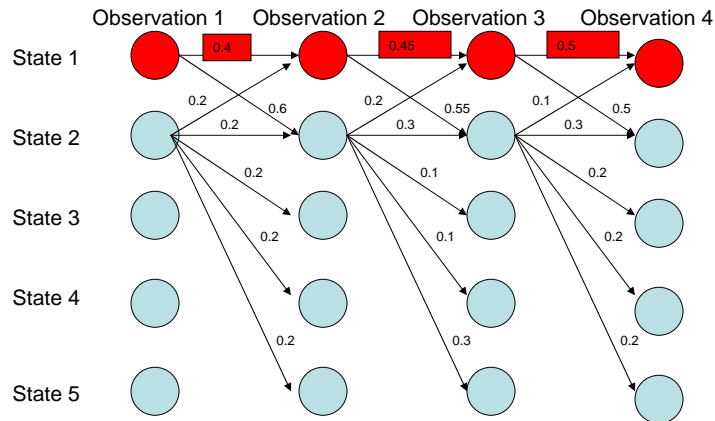
2->2->2->2: 0.018

1->2->1->2: 0.06

Eric Xing

24

MEMM: the Label bias problem



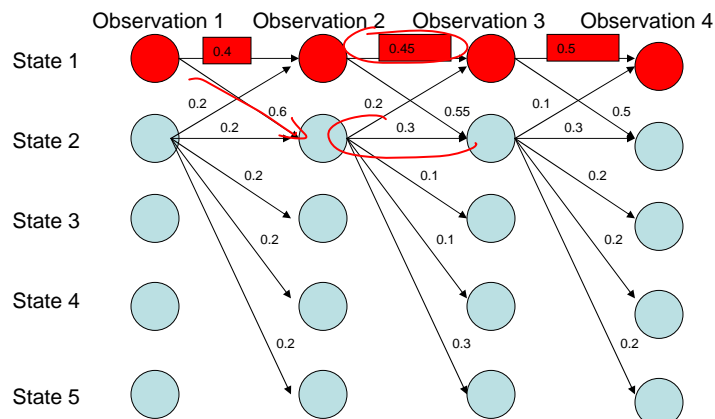
Most Likely Path: 1-> 1-> 1-> 1

- Although **locally** it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.
- **why?**

Eric Xing

25

MEMM: the Label bias problem



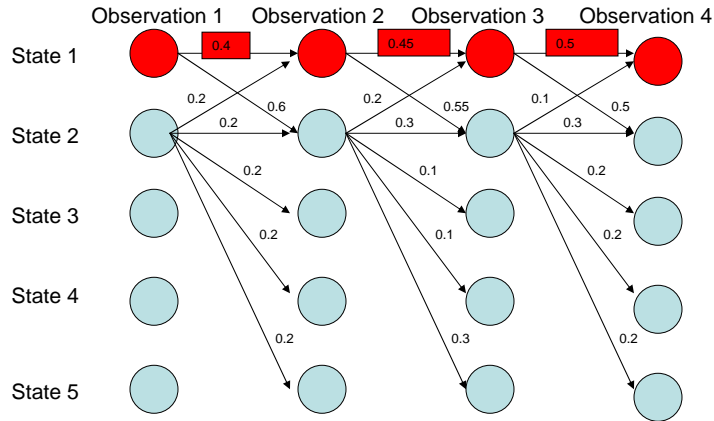
Most Likely Path: 1-> 1-> 1-> 1

- State 1 has only two transitions but state 2 has 5:
- Average transition probability from state 2 is lower

Eric Xing

26

MEMM: the Label bias problem



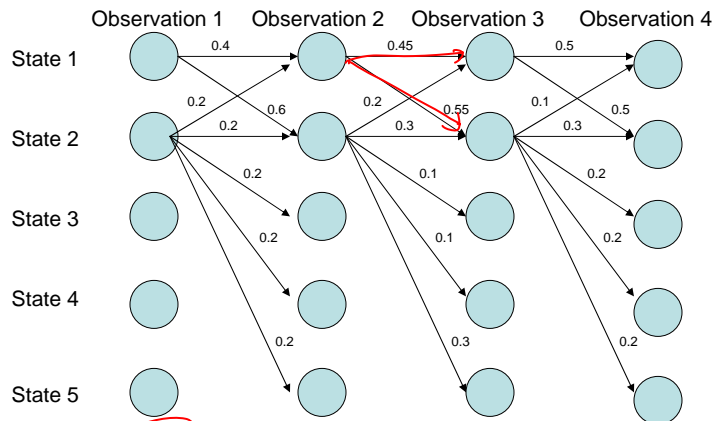
Label bias problem in MEMM:

- Preference of states with lower number of transitions over others

Eric Xing

27

Solution: Do not normalize probabilities locally

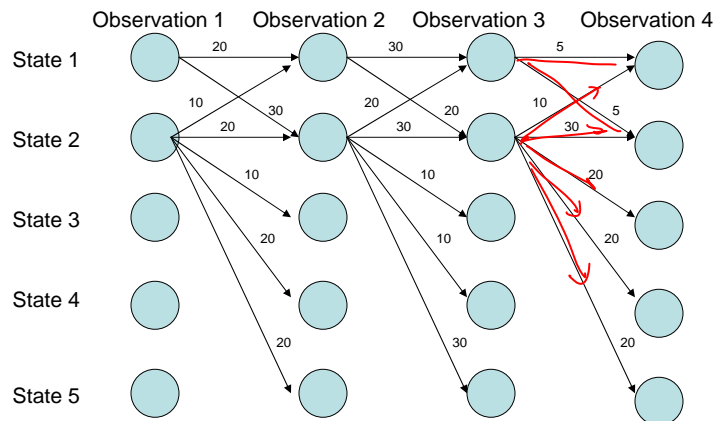


From local probabilities ...

Eric Xing

28

Solution: Do not normalize probabilities locally



From local probabilities to local potentials

- States with lower transitions do not have an unfair advantage!

Eric Xing

29



- Office hour

1 - 2 pm Mon

- Mid-term project milestone due today

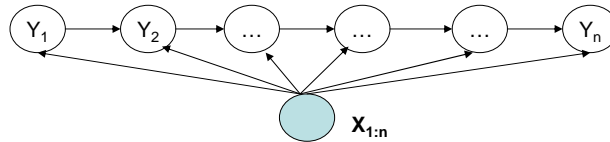
- Next week

4:30 - 5:30 this Friday

Eric Xing

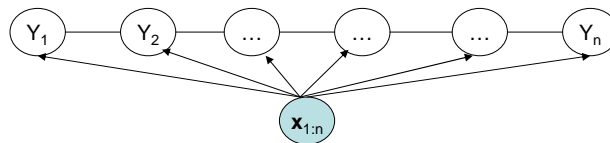
30

From MEMM



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

From MEMM to CRF



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

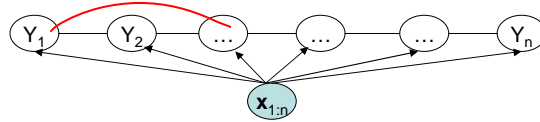
- CRF is a partially directed model
 - Discriminative model like MEMM
 - Usage of global normalizer $Z(\mathbf{x})$ overcomes the label bias problem of MEMM
 - Models the dependence between each state and the entire observation sequence (like MEMM)

Conditional Random Fields



- General parametric form:

$$\phi(\eta_i, \eta_{i-1}, \mathbf{x})$$



$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x}, \lambda, \mu)} \exp\left(\sum_{i=1}^n \left(\sum_k \lambda_k f_k(y_i, y_{i-1}, \mathbf{x}) + \sum_l \mu_l g_l(y_i, \mathbf{x})\right)\right)$$

$$= \frac{1}{Z(\mathbf{x}, \lambda, \mu)} \exp\left(\sum_{i=1}^n (\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}) + \mu^T \mathbf{g}(y_i, \mathbf{x}))\right)$$

where $Z(\mathbf{x}, \lambda, \mu) = \sum_{\mathbf{y}} \exp\left(\sum_{i=1}^n (\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}) + \mu^T \mathbf{g}(y_i, \mathbf{x}))\right)$

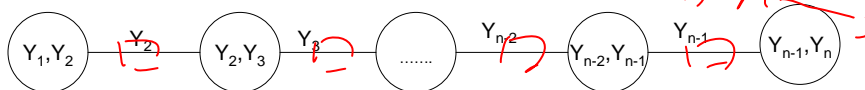
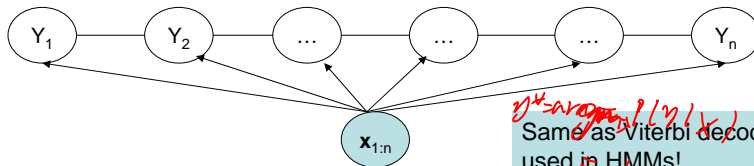
CRFs: Inference



- Given CRF parameters λ and μ , find the \mathbf{y}^* that maximizes $P(\mathbf{y}|\mathbf{x})$

$$\mathbf{y}^* = \arg \max_{\mathbf{y}} \exp\left(\sum_{i=1}^n (\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}) + \mu^T \mathbf{g}(y_i, \mathbf{x}))\right)$$

- Can ignore $Z(\mathbf{x})$ because it is not a function of \mathbf{y}
- Run the max-product algorithm on the junction-tree of CRF:



CRF learning

$\log z = \log \sum_y P(y|x)$
 $\frac{d}{d\lambda} \log z = \frac{1}{z} \frac{dz}{d\lambda} = \frac{1}{z} \sum_x \frac{d}{d\lambda} P(x)$

- Given $\{(\mathbf{x}_d, \mathbf{y}_d)\}_{d=1}^N$, find λ^*, μ^* such that

$$\lambda^*, \mu^* = \arg \max_{\lambda, \mu} L(\lambda, \mu) = \arg \max_{\lambda, \mu} \prod_{d=1}^N P(\mathbf{y}_d | \mathbf{x}_d, \lambda, \mu)$$

$$= \arg \max_{\lambda, \mu} \prod_{d=1}^N \frac{1}{Z(\mathbf{x}_d, \lambda, \mu)} \exp\left(\sum_{i=1}^n (\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d))\right)$$

$\log \Rightarrow \arg \max_{\lambda, \mu} \sum_{d=1}^N \left(\sum_{i=1}^n (\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d)) - \log Z(\mathbf{x}_d, \lambda, \mu) \right)$

$= \frac{1}{z} \sum_x P(x) \frac{d}{d\lambda} \exp(x)$
 $= P(y|x) \sum f(\cdot) \stackrel{\lambda}{=} \exp(x) \frac{dx}{d\lambda}$

- Computing the gradient w.r.t λ :

Gradient of the log-partition function in an exponential family is the expectation of the sufficient statistics.

$$\nabla_{\lambda} L(\lambda, \mu) = \sum_{d=1}^N \left(\sum_{i=1}^n \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}} (P(\mathbf{y} | \mathbf{x}_d) \sum_{i=1}^n \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d)) \right)$$

CRF learning

$$\nabla_{\lambda} L(\lambda, \mu) = \sum_{d=1}^N \left(\sum_{i=1}^n \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}} (P(\mathbf{y} | \mathbf{x}_d) \sum_{i=1}^n \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d)) \right)$$

- Computing the model expectations:

- Requires exponentially large number of summations: Is it intractable?

$$\sum_{\mathbf{y}} (P(\mathbf{y} | \mathbf{x}_d) \sum_{i=1}^n \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d)) = \sum_{i=1}^n \left(\sum_{\mathbf{y}} \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) P(\mathbf{y} | \mathbf{x}_d) \right)$$

$$= \sum_{i=1}^n \sum_{y_i, y_{i-1}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) P(y_i, y_{i-1} | \mathbf{x}_d)$$

Expectation of \mathbf{f} over the corresponding marginal probability of neighboring nodes!!

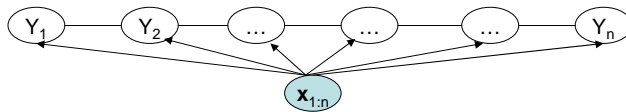
- Tractable!

- Can compute marginals using the sum-product algorithm on the chain

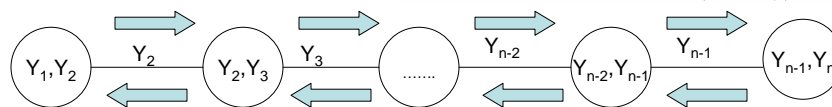
CRF learning



- Computing marginals using junction-tree calibration:



- Junction Tree Initialization: $\alpha^0(y_i, y_{i-1}) = \exp(\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_i, \mathbf{x}_d))$



- After calibration:

$$P(y_i, y_{i-1} | \mathbf{x}_d) \propto \alpha(y_i, y_{i-1})$$

$$\Rightarrow P(y_i, y_{i-1} | \mathbf{x}_d) = \frac{\alpha(y_i, y_{i-1})}{\sum_{y_i, y_{i-1}} \alpha(y_i, y_{i-1})} = \alpha'(y_i, y_{i-1})$$

Also called forward-backward algorithm

CRF learning



- Computing feature expectations using calibrated potentials:

$$\sum_{y_i, y_{i-1}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) P(y_i, y_{i-1} | \mathbf{x}_d) = \sum_{y_i, y_{i-1}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) \alpha'(y_i, y_{i-1})$$

- Now we know how to compute $r_\lambda L(\lambda, \mu)$:

$$\nabla_\lambda L(\lambda, \mu) = \sum_{d=1}^N \left(\sum_{i=1}^n \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}} (P(\mathbf{y} | \mathbf{x}_d) \sum_{i=1}^n \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)) \right)$$

$$= \sum_{d=1}^N \left(\sum_{i=1}^n (\mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{y_i, y_{i-1}} \alpha'(y_i, y_{i-1}) \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)) \right)$$

- Learning can now be done using gradient ascent:

$$\lambda^{(t+1)} = \lambda^{(t)} + \eta \nabla_\lambda L(\lambda^{(t)}, \mu^{(t)})$$

$$\mu^{(t+1)} = \mu^{(t)} + \eta \nabla_\mu L(\lambda^{(t)}, \mu^{(t)})$$

CRF learning



- In practice, we use a Gaussian Regularizer for the parameter vector to improve generalizability

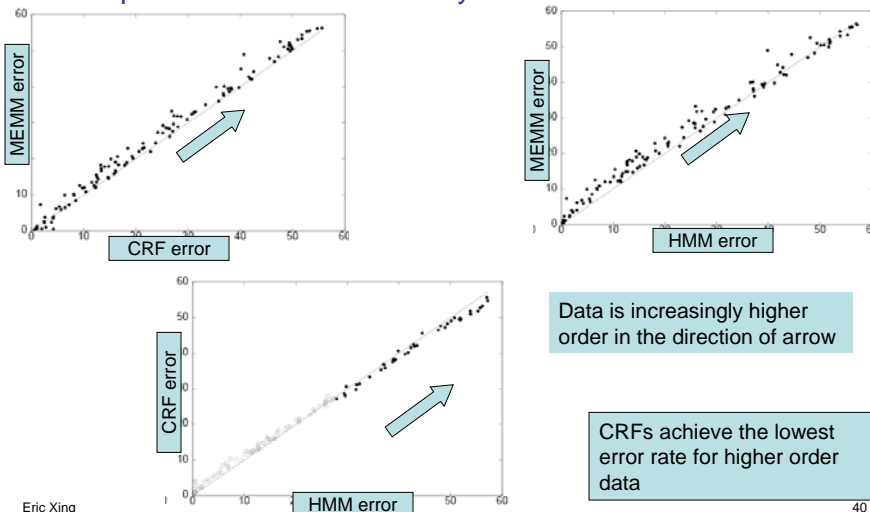
$$\lambda^*, \mu^* = \arg \max_{\lambda, \mu} \sum_{d=1}^N \log P(\mathbf{y}_d | \mathbf{x}_d, \lambda, \mu) - \frac{1}{2\sigma^2} (\lambda^T \lambda + \mu^T \mu)$$

- In practice, gradient ascent has very slow convergence
 - Alternatives:
 - Conjugate Gradient method
 - Limited Memory Quasi-Newton Methods

CRFs: some empirical results



- Comparison of error rates on synthetic data



CRFs: some empirical results



- Parts of Speech tagging

<i>model</i>	<i>error</i>	<i>oov error</i>
HMM	5.69%	45.99%
MEMM	6.37%	54.61%
CRF	5.55%	48.05%
MEMM ⁺	4.81%	26.99%
CRF ⁺	4.27%	23.76%

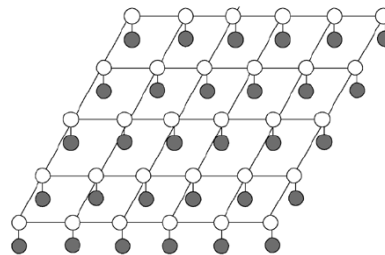
⁺Using spelling features

- Using same set of features: HMM >= CRF > MEMM
- Using additional overlapping features: CRF⁺ > MEMM⁺ >> HMM

Other CRFs



- So far we have discussed only 1-dimensional chain CRFs
 - Inference and learning: exact
- We could also have CRFs for arbitrary graph structure
 - E.g: Grid CRFs
 - Inference and learning no longer tractable
 - Approximate techniques used
 - MCMC Sampling
 - Variational Inference
 - Loopy Belief Propagation
 - We will discuss these techniques in the future



Summary



- Conditional Random Fields are partially directed discriminative models
- They overcome the label bias problem of MEMMs by using a global normalizer
- Inference for 1-D chain CRFs is exact
 - Same as Max-product or Viterbi decoding
- Learning also is exact
 - globally optimum parameters can be learned
 - Requires using sum-product or forward-backward algorithm
- CRFs involving arbitrary graph structure are intractable in general
 - E.g.: Grid CRFs
 - Inference and learning require approximation techniques
 - MCMC sampling
 - Variational methods
 - Loopy BP