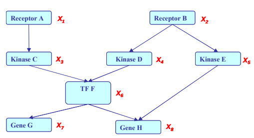
School of Computer Science
Carnegie Mellon

Hidden Markov Model and Conditional Random Fields

Probabilistic Graphical Models (10-708)

Lecture 12, Oct 29, 2007

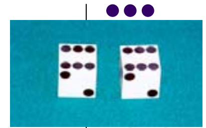


Eric Xing

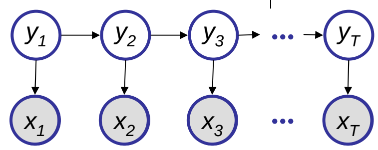
Reading: J-Chap. 12, and addition papers

- Feedbacks
  - Today
  - Office hour
- Recitation
- Project milestone
  - This Wednesday

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

- Start probabilities

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

- Emission probabilities associated with each state

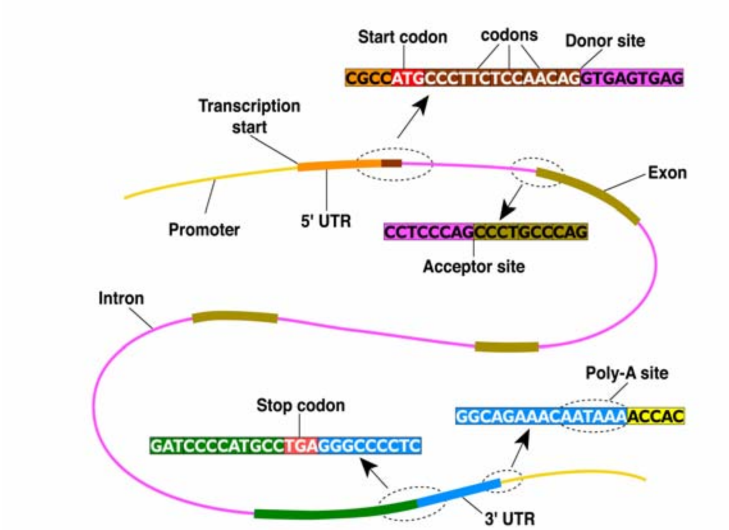
$$p(x_t \mid y_t^i = 1) \sim \text{Multinomial}\left(b_{i,1}, b_{i,2}, \ldots, b_{i,K}\right), \forall i \in \mathbb{I}.$$

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

Applications of HMMs

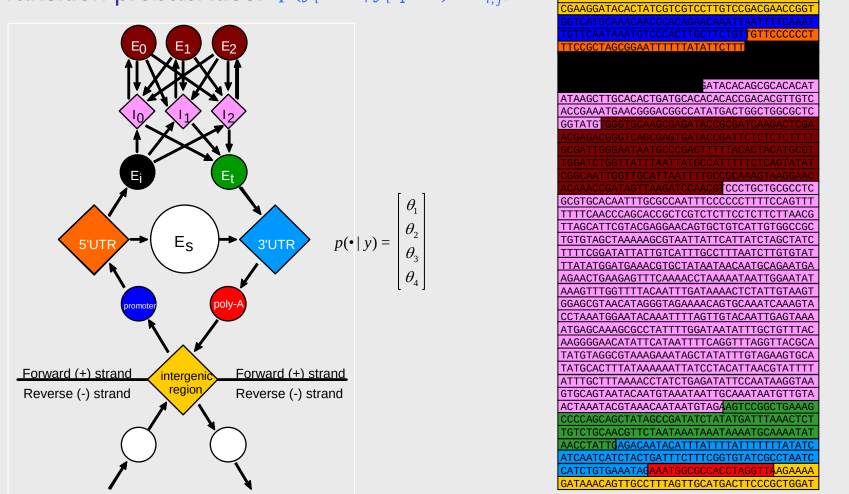
- **Some early applications of HMMs**
  - finance, but we never saw them
  - speech recognition
  - modelling ion channels
- In the mid-late 1980s HMMs entered genetics and molecular biology, and they are now firmly entrenched.
- **Some current applications of HMMs to biology**
  - mapping chromosomes
  - aligning biological sequences
  - predicting sequence structure
  - inferring evolutionary relationships
  - finding genes in DNA sequence

Typical structure of a gene



GENSCAN (Burge & Karlin)

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



$$p(\bullet \mid y) = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$

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 $\mathbf{x}$:

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

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

$$\alpha(y_t^k = 1) = \alpha_t^k \stackrel{\text{def}}{=} P(x_1,\ldots,x_t, y_t^k = 1)$$

(the forward probability)

  - The recursion:

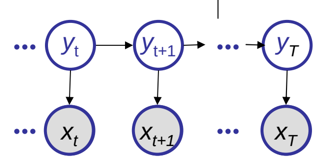
$$\alpha_t^k = p(x_t \mid y_t^k = 1)\sum_i \alpha_{t-1}^i a_{i,k}$$

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

The Backward Algorithm

- We want to compute $P(y_t^k = 1 \mid \mathbf{x})$,

  the posterior probability distribution on the t th position, given $\mathbf{x}$



  - We start by computing

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

P(x)

$$= P(x_1,\ldots,x_t, y_t^k = 1)P(x_{t+1},\ldots,x_T \mid x_1,\ldots,x_t, y_t^k = 1)$$

$$= \underbrace{P(x_1\ldots x_t, y_t^k = 1)}_{\text{Forward, } \alpha_t^k}\underbrace{P(x_{t+1}\ldots x_T \mid y_t^k = 1)}_{\text{Backward, } \beta_t^k = P(x_{t+1},\ldots,x_T \mid y_t^k = 1)}$$

0 t T

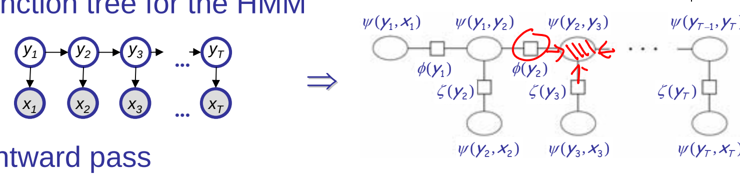
Forward, α_t^k **Backward,** $\beta_t^k = P(x_{t+1},\ldots,x_T \mid y_t^k = 1)$

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

The Junction tree algorithm

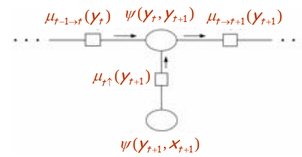
$\phi(x_c)$
$= \phi'(x_c) \prod M_{\to x_c}$

- A junction tree for the HMM



- Rightward pass

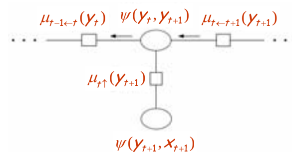
$$\mu_{t\to t+1}(y_{t+1}) = \sum_{y_t} \psi(y_t, y_{t+1}) \mu_{t-1\to t}(y_t) \mu_{t\uparrow}(y_{t+1})$$
$$= \sum_{y_t} p(y_{t+1} \mid y_t) \mu_{t-1\to t}(y_t) p(x_{t+1} \mid y_{t+1})$$
$$= p(x_{t+1} \mid y_{t+1}) \sum_{y_t} a_{y_t, y_{t+1}} \mu_{t-1\to t}(y_t)$$



  - This is exactly the *forward algorithm*!

- Leftward pass …

$$\mu_{t-1\leftarrow t}(y_t) = \sum_{y_{t+1}} \psi(y_t, y_{t+1}) \mu_{t\leftarrow t+1}(y_{t+1}) \mu_{t\uparrow}(y_{t+1})$$
$$= \sum_{y_{t+1}} p(y_{t+1} \mid y_t) \mu_{t\leftarrow t+1}(y_{t+1}) p(x_{t+1} \mid y_{t+1})$$



  - This is exactly the *backward algorithm*!

Eric Xing 9

Summary of the F-B algorithm

$$\alpha_t^k \stackrel{\text{def}}{=} \mu_{t-1\to t}(k) = P(x_1, \ldots, x_{t-1}, x_t, y_t^k = 1)$$
$$\alpha_t^k = p(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}$$
$$\beta_t^k \stackrel{\text{def}}{=} \mu_{t-1\leftarrow t}(k) = P(x_{t+1}, \ldots, x_T \mid y_t^k = 1)$$
$$\beta_t^k = \sum_i a_{k,i} p(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i$$

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

$$\xi_t^{i,j} \stackrel{\text{def}}{=} p(y_t^i = 1, y_{t+1}^j = 1, x_{1:T})$$
$$\propto \mu_{t-1\to t}(y_t^i = 1) \mu_{t\leftarrow t+1}(y_{t+1}^j = 1) p(x_{t+1} \mid y_{t+1}) p(y_{t+1} \mid y_t)$$
$$\xi_t^{i,j} = \alpha_t^i \beta_{t+1}^j a_{i,j} p(x_{t+1} \mid y_{t+1}^i = 1)$$

The matrix-vector form:

$$B_t(i) \stackrel{\text{def}}{=} p(x_t \mid y_t^i = 1) \quad B = (\ )$$
$$A(i,j) \stackrel{\text{def}}{=} p(y_{t+1}^j = 1 \mid y_t^i = 1) \quad A = \boxed{\ }$$

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

$p(y) \propto$
$\boxed{y}$
$\langle y \rangle_t \quad y_{t+1}$
$p(y_t, y_{t+1} \mid x)$

Eric Xing 10

Posterior decoding

y

x t

- We can now calculate

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

x

y_t

- Then, we can ask
  - What is the most likely state at position t of sequence $\mathbf{x}$:

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

  - Note that this is an MPA of a single hidden state, what if we want to a MPA of a whole hidden state sequence?
  - Posterior Decoding: $\left\{ y_t^{k_t^*} = 1 : t = 1 \cdots T \right\}$
  - 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 $\mathbf{x} = x_1, \ldots, x_T$, we want to find $\mathbf{y} = y_1, \ldots, y_T$, such that $P(\mathbf{y}|\mathbf{x})$ is maximized:

$$\mathbf{y}^* = \text{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \text{argmax}_{\pi} P(\mathbf{y},\mathbf{x})$$

$V_T^k = \max_{y_1 \cdots y_{T-1}} P(x_{1:T}, y_{1:T-1}, y_T^k = 1)$

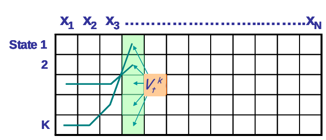
  - Let

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

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

  - The recursion:

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



  - Underflows are a significant problem

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

$t-2$ $t-1$ t

V_{t-1}^i

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

The Viterbi Algorithm – derivation

- Define the viterbi probability:

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

Computational Complexity and implementation details

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

$$\alpha_t^k = p(x_t \mid y_t^k=1)\sum_i \alpha_{t-1}^i a_{i,k}$$
$$\beta_t^k = \sum_i a_{k,i}\, p(x_{t+1} \mid y_{t+1}^i=1)\beta_{t+1}^i$$
$$V_t^k = p(x_t \mid y_t^k=1)\max_i a_{i,k}V_{t-1}^i$$

Time: $O(K^2N)$; Space: $O(KN)$.

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

Learning HMM

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

    GIVEN: a genomic region $x = x_1 \ldots x_{1,000,000}$ 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:**

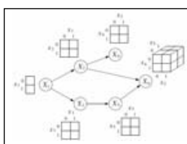
    GIVEN: the porcupine genome; we don’t know how frequent are the CpG islands there, neither do we know their composition

    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

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

- EM
  - The E step

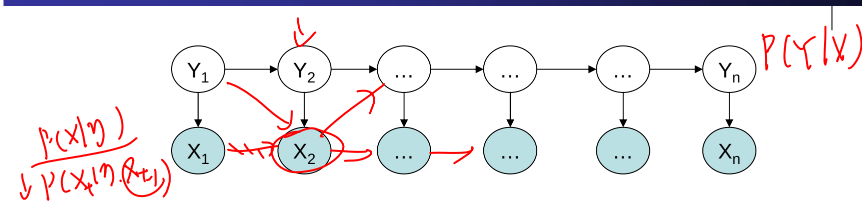
$$\gamma_{n,t}^i = \left\langle y_{n,t}^i \right\rangle = p(y_{n,t}^i = 1 \mid \mathbf{x}_n)$$

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

  - 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



- 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

  P(X, Y) → P(X) → P(Y|X) = P(X,Y)/P(X)

  - 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**)

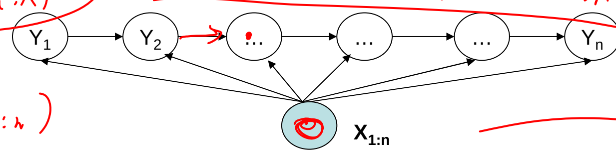
Solution:

Maximum Entropy Markov Model (MEMM)

$\backslash P(y, x)$ $\boxed{P(y|x)}$ $P(x)$

$P(x, y) = P(x) P(y|x) = P(x) \prod P(y_i | y_{i-1}, x)$

$P(y_i | y_{i-1}, x)$



$P(y_{1:n})$

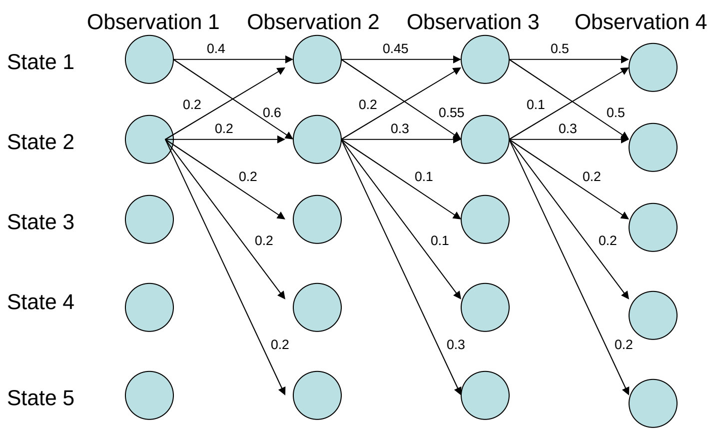
$P(y|x) = P(x,y) / P(x)$

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

$= \sum_{y_i} \exp(w f(y_i, y_{i-1}))$

- 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**)

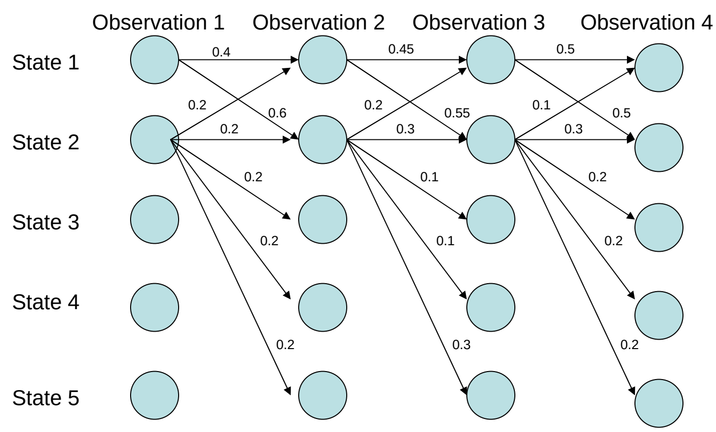
MEMM: the Label bias problem



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

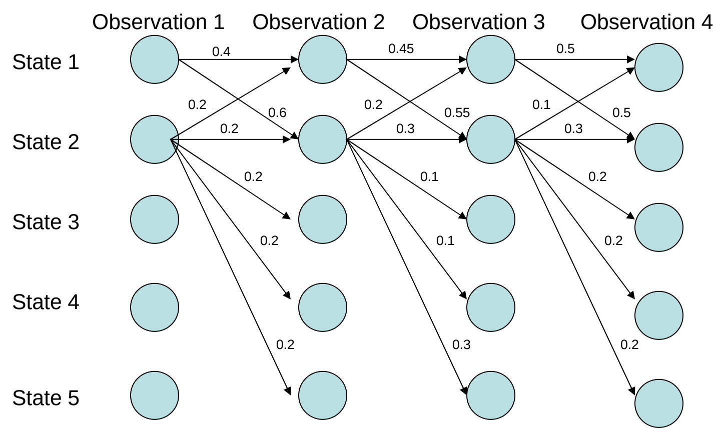
MEMM: the Label bias problem



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

- 0.4 x 0.45 x 0.5 = 0.09

MEMM: the Label bias problem

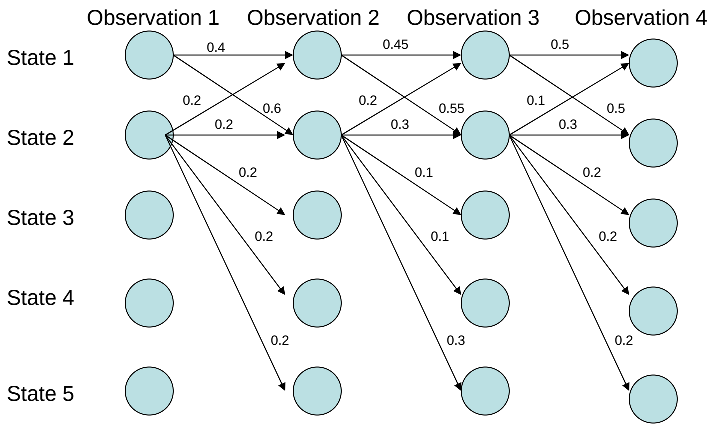


Probability of path 2->2->2->2 :

- 0.2 X 0.3 X 0.3 = 0.018

Other paths:
1-> 1-> 1-> 1: 0.09

MEMM: the Label bias problem



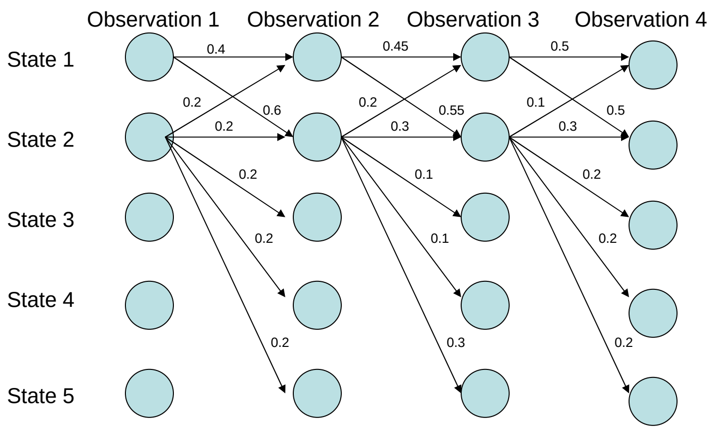
Probability of path 1->2->1->2:

- 0.6 X 0.2 X 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



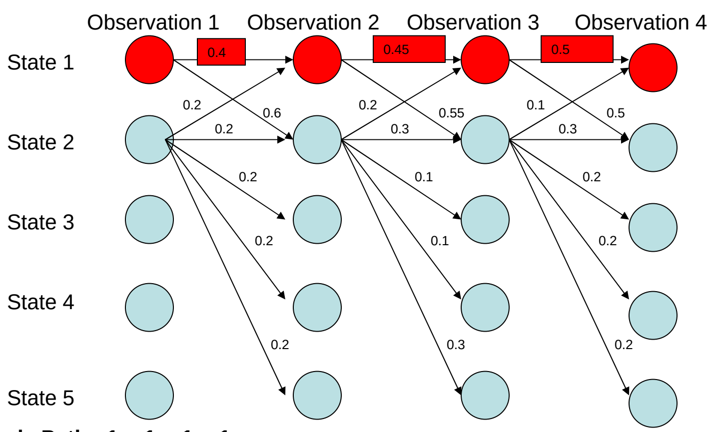
Probability of path 1->1->2->2:

- 0.4 X 0.55 X 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

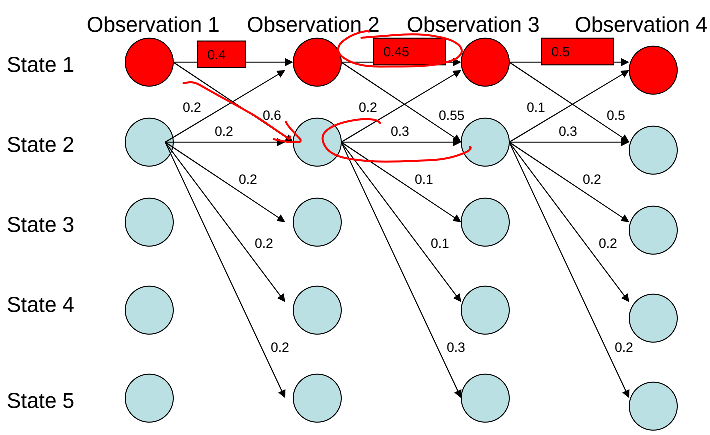
MEMM: the Label bias problem



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?**

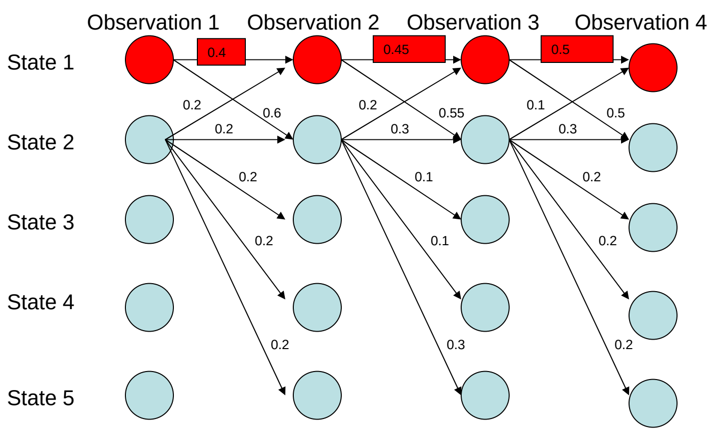
MEMM: the Label bias problem



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

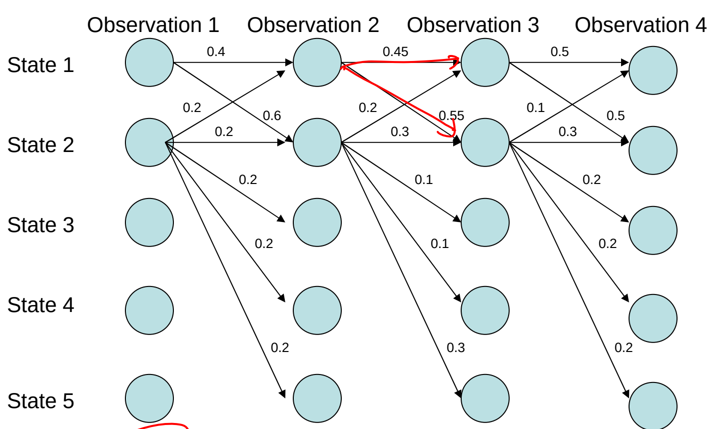
MEMM: the Label bias problem



Label bias problem in MEMM:

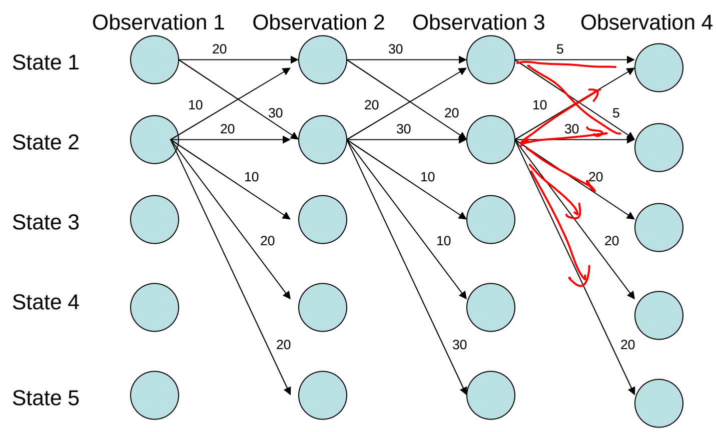
- Preference of states with lower number of transitions over others

Solution:
Do not normalize probabilities locally



From local probabilities ….

Solution:
Do not normalize probabilities locally



From local probabilities to local potentials

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

- Office hour

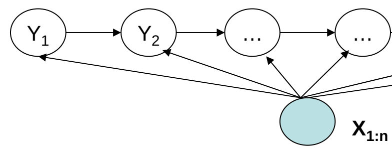
1-2 pm Mon

- Mid-term project milestone due today

- Next week

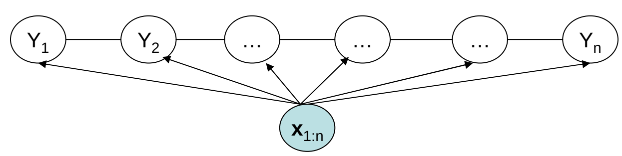
4:30 - 5:30 this Friday

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



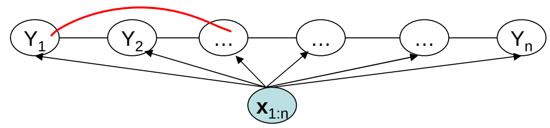
$$P(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n})}\prod_{i=1}^{n} \phi(y_i,y_{i-1},\mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n},\mathbf{w})}\prod_{i=1}^{n} \exp(\mathbf{w}^T\mathbf{f}(y_i,y_{i-1},\mathbf{x}_{1:n}))$$

- CRF is a partially directed model
  - Discriminative model like MEMM
  - Usage of global normalizer Z(**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:

$\phi(y_i, y_{i-1}, x)$



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

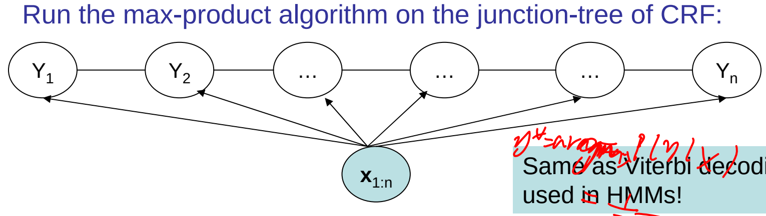
$$\text{where } Z(\mathbf{x},\lambda,\mu) = \sum_{\mathbf{y}}\exp\Big(\sum_{i=1}^{n}(\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}) + \mu^T \mathbf{g}(y_i, \mathbf{x}))\Big)$$

CRFs: Inference

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

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

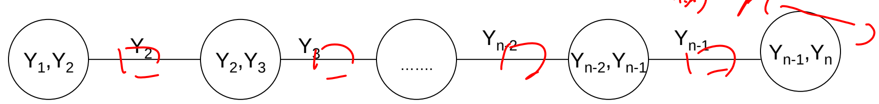
  - Can ignore Z($\mathbf{x}$) because it is not a function of $\mathbf{y}$
- Run the max-product algorithm on the junction-tree of CRF:



$y^* = \arg\max P(y|x)$

Same as Viterbi decoding used in HMMs!

$Z(x,\lambda,\mu)$ exp(



CRF learning

$\log Z = \log \sum_y (\tilde{p}(y|x))$

$\frac{d}{d\lambda} \log Z = \frac{1}{Z} \frac{dZ}{d\lambda} = \frac{1}{Z} \tilde{p} \frac{d}{d\lambda}$

$= \frac{1}{Z} \tilde{p} \sum f(y_i, y_{i-1})$

$= P(y|x) \sum f(\_)$

$\frac{d}{\lambda} \exp x = \exp x \frac{dx}{da}$

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

$$\begin{aligned}
\lambda*, \mu* &= \arg\max_{\lambda,\mu} L(\lambda, \mu) = \arg\max_{\lambda,\mu} \prod_{d=1}^{N} P(\mathbf{y}_d | \mathbf{x}_d, \lambda, \mu) \\
&= \arg\max_{\lambda,\mu} \prod_{d=1}^{N} \frac{1}{Z(\mathbf{x}_d, \lambda, \mu)} \exp(\sum_{i=1}^{n} (\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d))) \\
&= \arg\max_{\lambda,\mu} \sum_{d=1}^{N} (\sum_{i=1}^{n} (\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d)) - \log Z(\mathbf{x}_d, \lambda, \mu))
\end{aligned}$$

$\log \Rightarrow$

- Computing the gradient w.r.t λ:

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

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

CRF learning

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

- Computing the model expectations:
  - Requires exponentially large number of summations: Is it intractable?

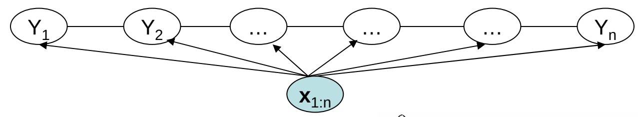
$$\begin{aligned}
\sum_{\mathbf{y}} (P(\mathbf{y}|\mathbf{x}_d) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)) &= \sum_{i=1}^{n} (\sum_{\mathbf{y}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) P(\mathbf{y}|\mathbf{x}_d)) \\
&= \sum_{i=1}^{n} \sum_{y_i, y_{i-1}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)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

Expectation of **f** over the corresponding marginal probability of neighboring nodes!!

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

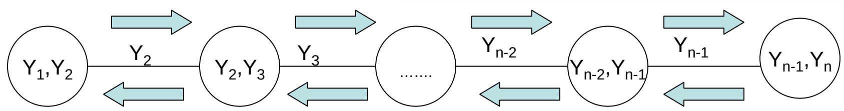
CRF learning

- Computing marginals using junction-tree calibration:



- Junction Tree Initialization:

$$\begin{aligned}\alpha^0(y_i, y_{i-1}) &= \exp(\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) \\ &\quad + \mu^T \mathbf{g}(y_i, \mathbf{x}_d))\end{aligned}$$



- After calibration:

Also called
forward-backward algorithm

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

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 $r_\lambda L(\lambda,\mu)$:

$$\begin{aligned}\nabla_\lambda L(\lambda, \mu) &= \sum_{d=1}^{N} (\sum_{i=1}^{n} \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}} (P(\mathbf{y}|\mathbf{x_d}) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d))) \\ &= \sum_{d=1}^{N} (\sum_{i=1}^{n} (\mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{y_i, y_{i-1}} \alpha'(y_i, y_{i-1}) \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)))\end{aligned}$$

- Learning can now be done using gradient ascent:

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

CRF learning

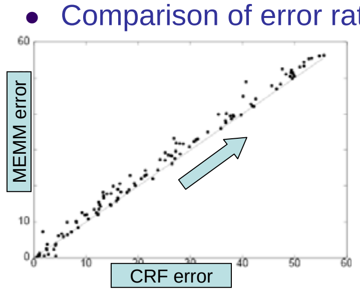
- In practice, we use a Gaussian Regularizer for the parameter vector to improve generalizability

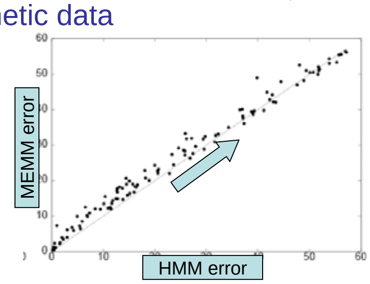
$$\lambda*, \mu* \quad = \quad \arg\max_{\lambda,\mu} \sum_{d=1}^{N} \log P(\mathbf{y}_d|\mathbf{x}_d, \lambda, \mu) - \frac{1}{2\sigma^2}(\lambda^T\lambda + \mu^T\mu)$$

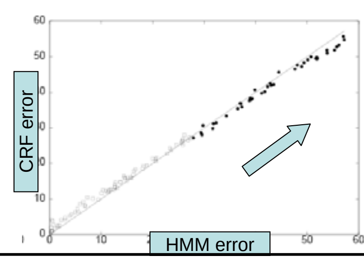
- In practice, gradient ascent has very slow convergence
  - Alternatives:
    - Conjugate Gradient method
    - Limited Memory Quasi-Newton Methods

CRFs: some empirical results

- Comparison of error rates on synthetic data







Data is increasingly higher order in the direction of arrow

CRFs achieve the lowest error rate for higher order data

CRFs: some empirical results

- Parts of Speech tagging

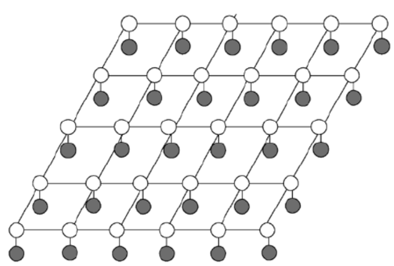
model	error	oov error
HMM	5.69%	45.99%
MEMM	6.37%	54.61%
CRF	5.55%	48.05%
MEMM+	4.81%	26.99%
CRF+	4.27%	23.76%

+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