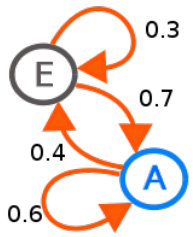
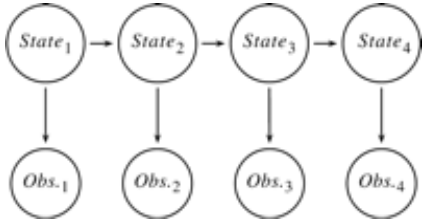
CSCI 599
Class Presentation

USC
UNIVERSITY
OF SOUTHERN
CALIFORNIA

Zach Levine

Markov Chain Monte Carlo (MCMC) HMM Parameter Estimates

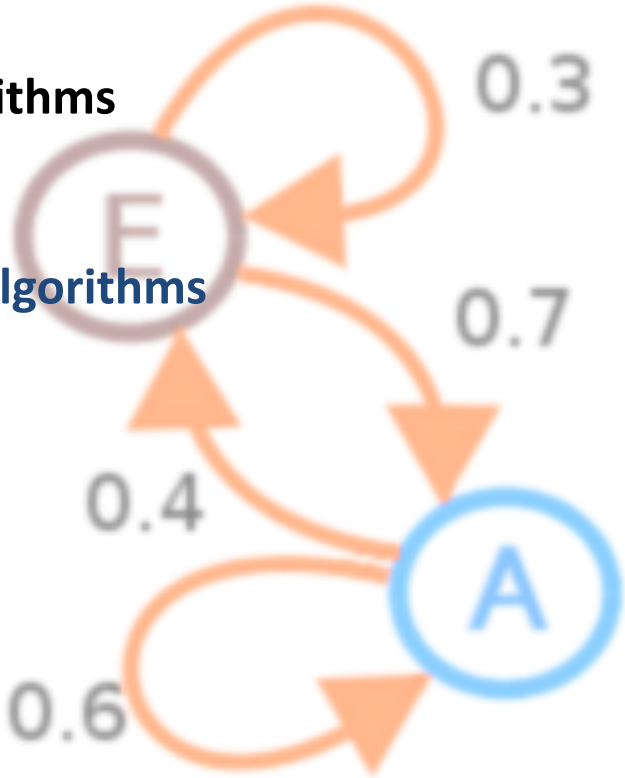




April 26th, 2012

Topics Covered in this Presentation

- A (Brief) Review of HMMs
- HMM Parameter Learning
  - Expectation-Maximization (EM) Algorithms
    - Baum-Welch Algorithm
    - Viterbi Learning Algorithm
  - Markov Chain Monte Carlo (MCMC) Algorithms
    - Metropolis-Hastings Algorithm
    - Gibbs Sampling Algorithm
- Comparisons of EM and MCMC models.
- Work in Progress
- Summary

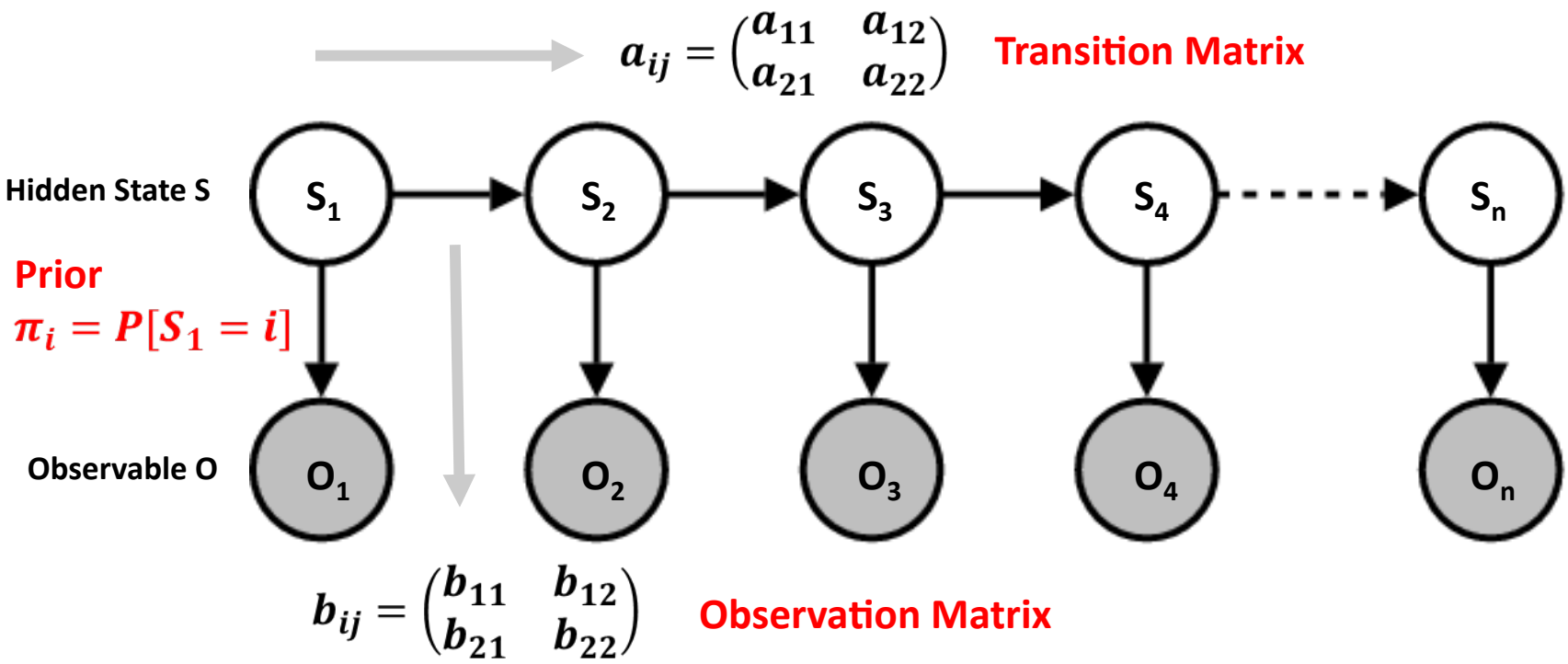


Question

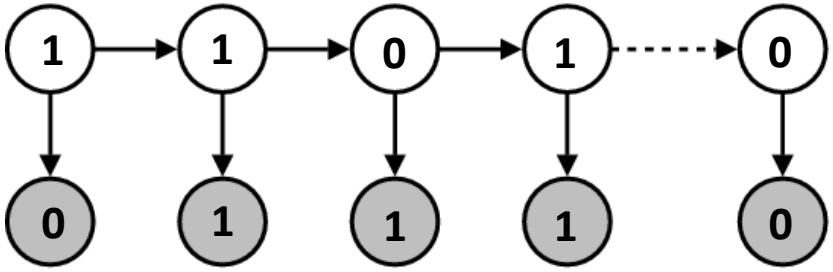
Given an HMM (a sequence of noisy observations which come from true hidden states), how do we infer the parameters from this model such that we maximize either the likelihood of the observations (Maximum Likelihood) or the likelihood of the most probable (posterior) sequence (Maximum a Posteriori).

Review of Hidden Markov Models (HMMs)

A simple binary HMM can be characterized by:



e.g.



For convenience, we can write the complete parameter set as:

$\lambda = (a_{ij}, b_{ij}, \pi_i)$

Review of Hidden Markov Models (HMMs)

Given λ, one can calculate the probability $P(O|\lambda)$ of some observed sequence $O = O_1O_2O_3\ldots O_N$. If we want to maximize the likelihood of the observed sequence O (i.e. **maximum likelihood**), we need λ to satisfy:

$$\lambda_{ML} = \text{argmax}_\lambda p_\lambda(O)$$

Or, if we want instead to maximize the likelihood of the most probable sequence (i.e. **maximum a posteriori**), then λ must satisfy:

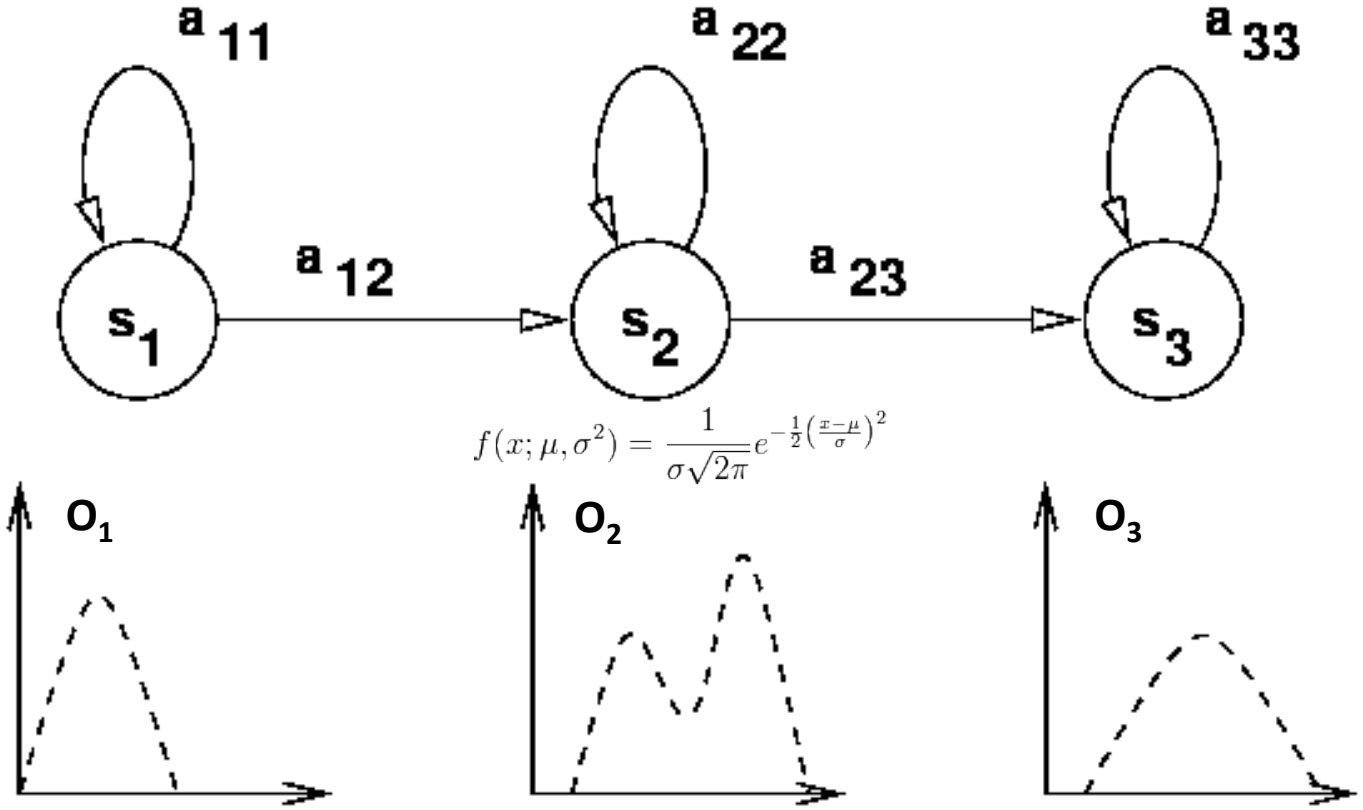
$$\lambda_{MAP} = \text{argmax}_\lambda \text{max}_s p_\lambda(S,O)$$

One can also estimate the hidden states $\{S\}$ from $P(\{S\}|O,\lambda)$ which can be deduced (for instance) from the Viterbi algorithm.

The question again is, how do we optimize λ?
Are some methods better than others?

Review of Hidden Markov Models (HMMs)

Also, can the models we use to deduce λ accommodate observations which follow continuous distributions?



HMM Parameter Learning

Methods of obtaining λ can be broken up into two broad categories

Frequentist Inference	Bayesian Inference
METHOD Frequently sample the observations of an HMM until some confidence interval is obtained for λ (e.g. aij = 95% of the true aij). Deterministic. Assumes that with enough sampling, a 'correct' λ will eventually be found.	**METHOD** Use Bayes' Rule on prior probabilities to continuously update the probability of the current state, e.g. the posterior. Furthermore the use of random numbers can allow for quick convergence when estimating λ.
EXAMPLES Expectation Maximization (EM) - Baum-Welch Algorithm	**EXAMPLES** Viterbi-Training Algorithm Markov-Chain Monte Carlo (MCMC) - Metropolis-Hastings - Gibbs Sampling

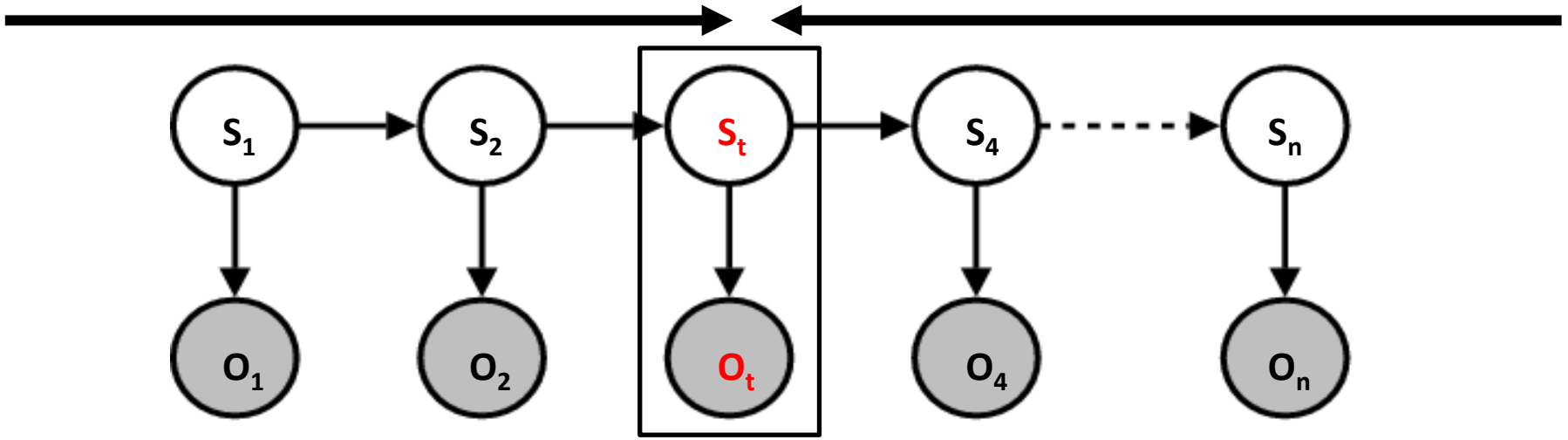
HMM Parameter Learning (EM)

EM is not per se a tool for frequentist (ML) inference, but a framework that can also be used for computing Bayesian (MAP) estimates. Thus it is considered a method for computing 'point estimates', or single-valued results.

Recall for instance that EM approaches use a '**forward-backward**' algorithm. That is it traverses the Markov-Chain forward and backwards until λ converges to some confidence interval.

Forward Probability $\alpha_{0:t}(i) = P(S_t = i, \{O\}_1^t | \pi)$

Backward Probability $\beta_{t+1:n}(i) = P(\{O\}_{t+1}^n | S_t = i)$



HMM Parameter Learning (EM)

The **Baum-Welch algorithm** was the first to implement this method and calculates these probabilities using:

Initial Conditions

$\alpha_1(i) = \pi_i b_i(O_1)$

$\beta_N(i) = 1$

and

Recursion Relationship

$\alpha_t(i) = P(S_t = i, \{O\}_1^t|\pi) = b_i(O_t)\sum_j a_{ji}\,\alpha_{t-1}(j)$

$\beta_t(i) = P(\{O\}_{t+1}^n|S_t = i) = \sum_j a_{ij} b_j(O_{t+1})\beta_{t+1}(j)$

Where a_{ij} = Transition Matix = $P(S_t=i|S_{t+1}=j)$
Where $b_i(O_t)$ = Probability of observing O_t when $S_t=i$. } λ

Baum-Welch then combines the entire observation history $\{O\}_1^n$ and the forward/backward variables to obtain new variables:

$$\xi_t(i,j) = P(S_t=i, S_{t+1}=j \mid \{O\}_1^n, \lambda) = \frac{\alpha_t(i)a_{ij}b_j(x_{t+1})\beta_{t+1}(j)}{\sum_i\sum_j \alpha_t(i)a_{ij}b_j(x_{t+1})\beta_{t+1}(j)}$$

$$\gamma_t(i) = P(S_t=i \mid \{O\}_1^n, \lambda) = \sum_j \xi_t(i,j) = \frac{\alpha_t(i)a_{ij}\beta_t(i)}{\sum_i \alpha_t(i)a_{ij}\beta_t(i)}$$

HMM Parameter Learning (EM)

The **Baum-Welch algorithm** was the first to implement this method and calculates these probabilities using:

Initial Conditions

$\alpha_1(i) = \pi_i b_i(O_1)$

$\beta_N(i) = 1$

and

Recursion Relationship

$\alpha_t(i) = P(S_t = i, \{O\}_1^t | \pi) = b_i(O_t) \sum_j a_{ji}\, \alpha_{t-1}(j)$

$\beta_t(i) = P(\{O\}_{t+1}^n | S_t = i) = \sum_j a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$

Where a_{ij} = Transition Matix = $P(S_t=i|S_{t+1}=j)$
Where $b_i(O_t)$ = Probability of observing O_t when $S_t=i$. } λ

New variables ξ_t and γ_t allow for the iterative calculation of $\lambda = (a_{ij}, b_i)$.

Make some initial guess for a_{ij} and b_i. Sample often. Obtain a convergent $\lambda = (a_{ij}, b_i, \pi)$ where $\pi_i = \gamma_1(i)$

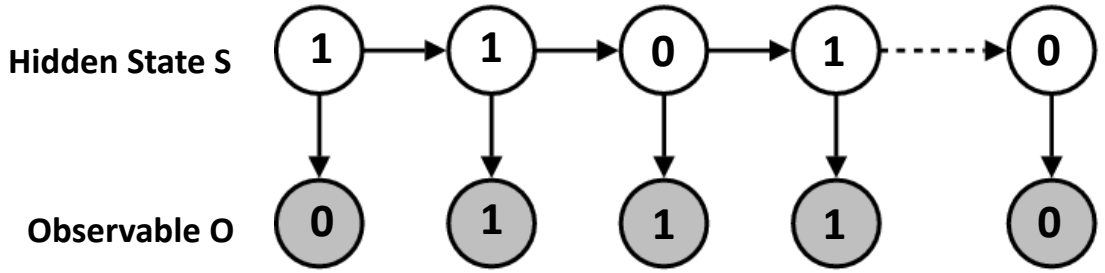
$$a_{ij} \leftarrow \frac{\sum_{t=1}^{N-1} \xi_t(i,j)}{\sum_{t=1}^{N-1} \gamma_t(i)}$$

$$b_i(k) \leftarrow \frac{\sum_{t=1}^{N} \gamma_t(i)\delta(O_t=k)}{\sum_{t=1}^{N} \gamma_t(i)}$$

Drawbacks: Potentially slow convergence (each iteration represents a full chain sweep), local maximization of data likelihood rather than global maximization, how big must N be to get some desired precision quickly?

HMM Parameter Learning (Bayesian)

Bayesian algorithms don't use MLE, but some algorithms such as the **Viterbi Learning algorithm** use MAP estimates instead which maximize the likelihood of the most likely hidden sequence. Essentially it returns a λ such that the observation sequence {O} minimizes some ground-state Hamiltonian H(O,S).



Consider the simple binary symmetric HMM where the probability that some observable is flipped from it's true state is ε, and the probability of flipping a bit from t to t+1 is q. Thus $\lambda = (q, \varepsilon)$

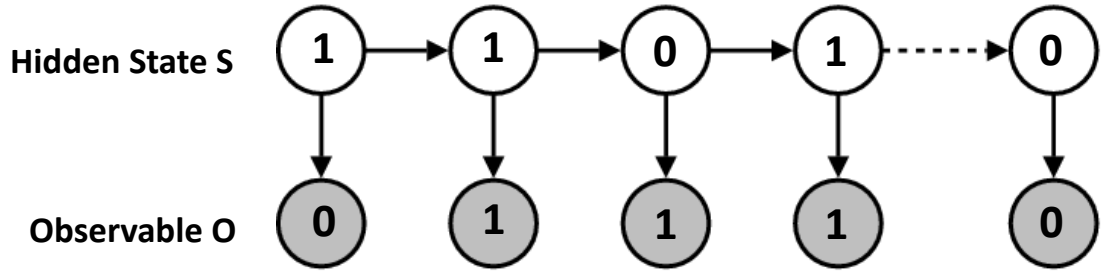
For a given Hamiltonian:

$H(x) = -J\sum_{k=1}^{N} S_k S_{k+1} - h\sum_{k=1}^{N} O_k S_k$ where $J = \frac{1}{2}\ln[\frac{1-q}{q}]$, $h = \frac{1}{2}\ln\left[\frac{1-\epsilon}{\epsilon}\right]$

one can obtain $\lambda^* = (q^*, \varepsilon^*) = \text{argmin}_{q,\varepsilon}[\min_S H(S|O; J, h)$

HMM Parameter Learning (Bayesian)

Bayesian algorithms don't use MLE, but some algorithms such as the Viterbi Learning algorithm use MAP estimates instead which maximize the likelihood of the most likely hidden sequence. Essentially it returns a λ such that the observation sequence {O} minimizes some ground-state Hamiltonian H(O,S).



Viterbi Algorithm

1.) Start with a trial q and ε.

2.) Find the MAP estimate

$$\hat{S} = argmax_s p(S|O;q,\varepsilon)$$

3.) Obtain a new q and ε set.

$$(\hat{q},\hat{\varepsilon}) = argmax_{q,\varepsilon} p(\hat{S}|O;q,\varepsilon)$$

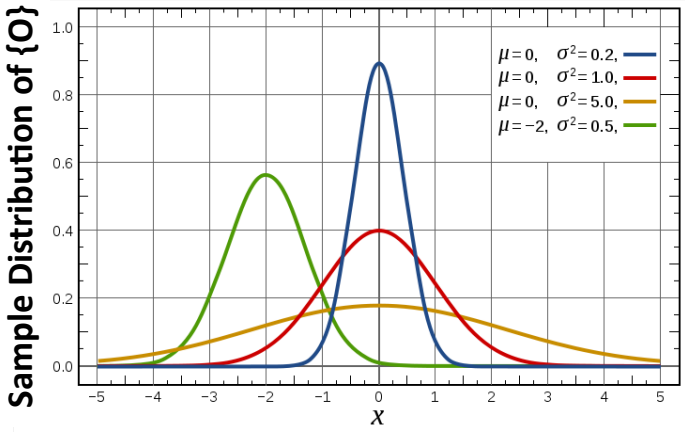
4.) Rinse and Repeat until λ = (q,ε) is convergent

Viterbi Learning is fast, but can however lead to inconsistent estimates compared to EM since Viterbi estimators need not be maximum likelihood estimators.

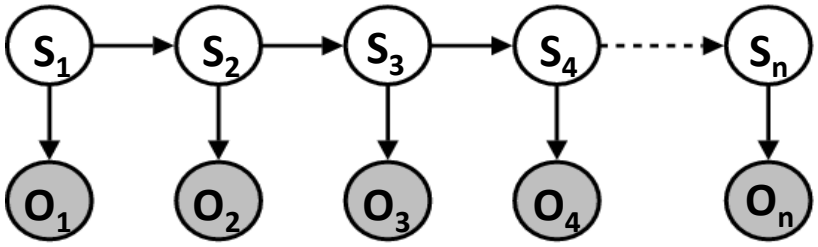
Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) algorithms are a class of Bayesian inference algorithms for sampling probability distributions by constructing a Markov chain (hidden states) that has some desired distribution as it's equilibrium distribution.

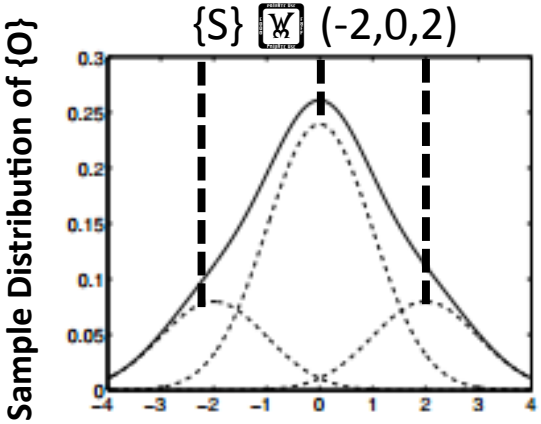
For example, suppose we sample observations {O} ∀ ∀ χ^N and we're given some standard distribution P (S_t) for each state (e.g. N(μ=0, σ^2=0.2)). The total mixed system ∀P(s)is very complex when N is large because there are many ways to construct it.



Markov Chain Monte Carlo methods can observe {O} and iteratively construct a Markov Chain {S} such that it's equilibrium distribution = ∀P(S).



Eventually the Markov Chain {S} will converge with distribution P(S), in which case $\lambda = (a_{ij}, b_i, \pi)$ can be extracted.



Markov Chain Monte Carlo

Many separate **MCMC random-walk algorithms** exists such as:

- Metropolis-Hastings Algorithm
- Gibbs Sampling Algorithm
- Multiple-Try Metropolis Algorithm

All of these methods try and sample the entire state-space in order to reproduce the equilibrium distribution.

For example: Metropolis-Hastings for a simple Markov Chain. Suppose we want to construct a Markov chain with a probability distribution P(S).

1.) Pick an arbitrary probability density $Q(S'|S_t)$ which suggests a new state S' from S_t.
Note that $Q(S'|S_t)$ must be symmetric (e.g. $Q(S'|S_t) = Q(S_t|S')$ such as a gaussian)
2.) For each state S_t, propose the next state a value S'_{t+1} which is generated from $Q(S'|S_t)$
3.) Calculate an acceptance ratio $a = P(S')/P(S_t)$
4.) If a ≥ 1, accept S' as S_{t+1}. Else, accept S' as S_{t+1} if rand(0,1) ≤ a. Else return to (2)
5.) With S_{t+1} updated, move forward and repeat steps (2-5).

Markov Chain Monte Carlo

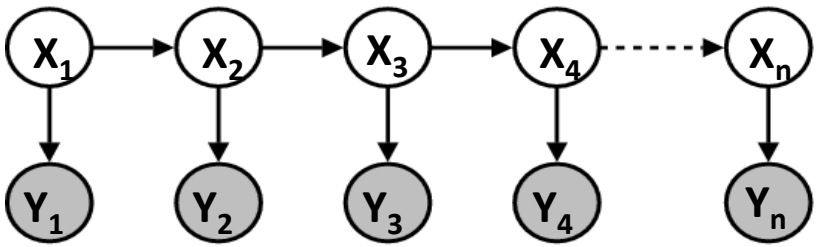
For Bayesian inference in an HMM, the **Gibbs Sampling Algorithm** is commonly used to extract $\lambda = (a_{ij}, b_i, \pi)$ which is a special case of Metropolis-Hastings for multivariate distributions. This procedure also allows us to extract credibility intervals for multiple parameters λ, rather than one (local) point estimate from EM.

For a simple HMM, studies have compared EM models to MCMC.

Bayesian Analysis (2008) **3**, Number 4, p. 659-688

EM versus Markov chain Monte Carlo for Estimation of Hidden Markov Models: A Computational Perspective

Tobias Rydén*



$$A = \{a_{ij}\} = \begin{pmatrix} 0.6 & 0.3 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.3 & 0.6 \end{pmatrix}$$

$$Y_k | X_k = i \sim \mathrm{N}(\mu_i, \sigma^2)$$

$\mu = (\mu_1, \mu_2, \mu_3) = (-2, 0, 2)$, and σ being either 0.5, 1 or 1.5. The stationary distribution of the hidden chain is (0.2, 0.6, 0.2) and the chain is assumed stationary.

Markov Chain Monte Carlo

Bayesian Analysis (2008) **3**, Number 4, p. 659-688

EM versus Markov chain Monte Carlo for Estimation of Hidden Markov Models: A Computational Perspective

Tobias Rydén*

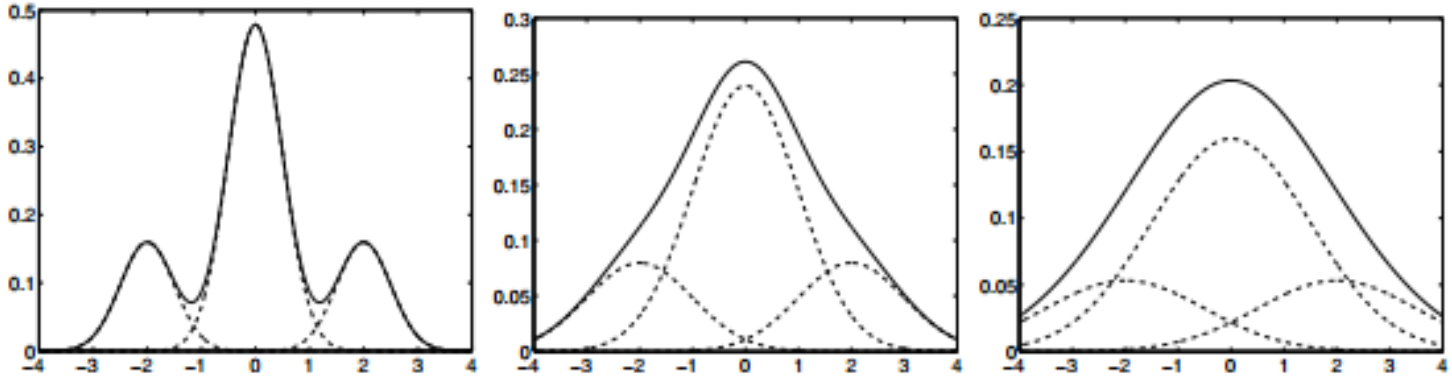


Figure 1: *Densities of the Normal components, weighted by the stationary probabilities (dashed lines) and marginal densities of the observations Y_k (solid lines) for the hidden Markov model of Case I with $\sigma = 0.5$ (left panel), $\sigma = 1$ (middle panel) and $\sigma = 1.5$ (right panel).*

Markov Chain Monte Carlo

The **Gibbs Sampling Algorithm** used in this paper is as follows:

Bayesian Analysis (2008) 3, Number 4, p. 659-688

EM versus Markov chain Monte Carlo for Estimation of Hidden Markov Models: A Computational Perspective

Tobias Rydén*

full conditional distributions are given by

$$(\rho_1, \rho_2, \ldots, \rho_d)|\ldots \sim \mathrm{Dir}(I\{X_1 = 1\} + 1, I\{X_1 = 2\} + 1, \ldots, I\{X_1 = d\} + 1) \quad (1)$$

where $I\{\cdot\}$ denotes an indicator function,

$$(a_{i1}, a_{i2}, \ldots, a_{id})|\ldots \sim \mathrm{Dir}(n_{i1} + 1, n_{i2} + 1, \ldots, n_{id} + 1) \quad (2)$$

where $n_{ij} = \#\{1 < k \leq n : X_{k-1} = i, X_k = j\}$ is the number of transitions from state i to j in the latent state sequence and with conditional independence across rows $i = 1, 2, \ldots, d$,

$$\mu_i|\ldots \sim \mathrm{N}\left(\frac{S_i + \kappa\xi\sigma^2}{n_i + \kappa\sigma^2}, \frac{\sigma^2}{n_i + \kappa\sigma^2}\right) \quad (3)$$

where $S_i = \sum_{k:\, X_k = i} y_k$, $n_i = \#\{1 \leq k \leq n : X_k = i\}$ is the number of visits to state i in the latent state sequence and with conditional independence across $i = 1, 2, \ldots, d$,

$$\xi = (\min y_k + \max y_k)/2 \text{ and } \kappa = 1/R^2 \text{ where } R = \max y_k - \min y_k$$

$$\sigma^{-2}|\ldots \sim \Gamma\left(\alpha + \frac{1}{2}n, \beta + \frac{1}{2}\sum_{k=1}^{n}(y_k - \mu_{X_k})^2\right), \quad (4)$$

and

$$\beta|\ldots \sim \Gamma(g + \alpha, h + \sigma^{-2}). \quad (5)$$

Here in all cases '...' denotes other parameters, the latent Markov chain and the data. Moreover, for the latent chain it holds that given parameters and data, this process is a non-homogeneous Markov chain with initial distribution

$$\mathbf{P}(X_1 = j \,|\, \ldots) \propto \rho_j \, \varphi(y_1; \mu_j, \sigma^2) \, p_\theta(\mathbf{y}_{2:n}|X_1 = j) \quad (6)$$

and transition probabilities

Backward Probabilities

$$\mathbf{P}(X_k = j \,|\, X_{k-1} = i) \propto a_{ij} \, \varphi(y_k; \mu_j, \sigma^2) \, p_\theta(\mathbf{y}_{k+1:n}|X_k = j) \quad (7)$$

where φ is the density of a Normal distribution with the indicated mean and variance.

Markov Chain Monte Carlo

The **Gibbs Sampling Algorithm** used in this paper is as follows:

Bayesian Analysis (2008) 3, Number 4, p. 659-688

EM versus Markov chain Monte Carlo for Estimation of Hidden Markov Models: A Computational Perspective

Tobias Rydén*

The full Gibbs sampler then amounts to alternating between updating the parameters conditional on the data and hidden Markov chain, and updating the hidden chain conditional on the data and parameters. In our implementation, this was done in the following order.

(a1) Update $(\mu_1, \ldots, \mu_d)$ by drawing independently from (3).

(a2) Update σ^2 by drawing from (4).

(a3) Update β by drawing from (5).

(a4) Update A by drawing $(a_{i1}, \ldots, a_{id})$ from (2), independently for $i = 1, \ldots, d$.

(a5) Update $(\rho_1, \ldots, \rho_d)$ by drawing from (1).

(b) Update $\{X_k\}_{k=1}^n$ by drawing X_1 from (6) and then X_k from (7) for $k = 2, 3, \ldots, n$.

One sequence of these steps (a) and (b) is typically referred to as a *sweep* of the Gibbs sampler.

Comparisons between EM and MCMC

For a Markov Chain of n = 1000 for three different values of σ^2, the authors found:

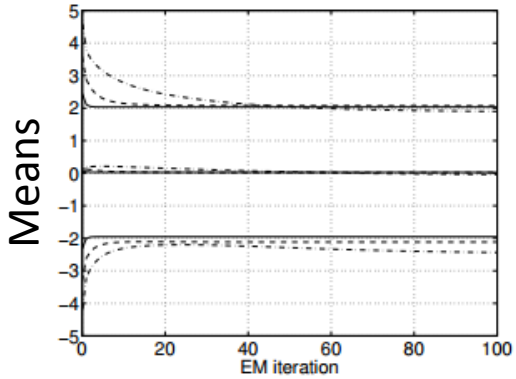
Bayesian Analysis (2008) 3, Number 4, p. 659-688

EM versus Markov chain Monte Carlo for Estimation of Hidden Markov Models: A Computational Perspective

Tobias Rydén*

EM

σ^2 = 0.5,1,2.5



Comparisons

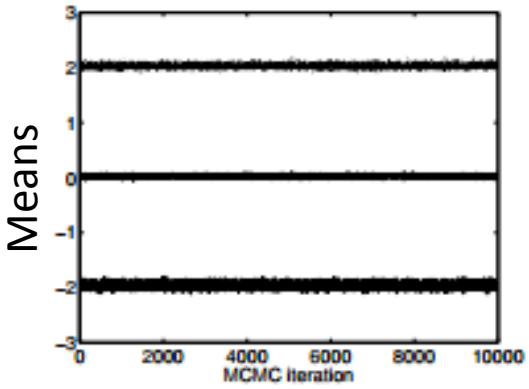
MCMC is faster. To reach 95% confidence of the true mean values (and thus a_{ij}), EM took 2177 sec to complete whereas MCMC took 237 sec.

MCMC can be noisy when the variance is high

EM may not always obtain the global solution

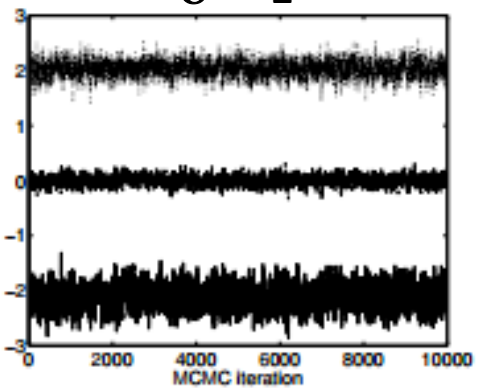
MCMC

σ^2 = 0.5



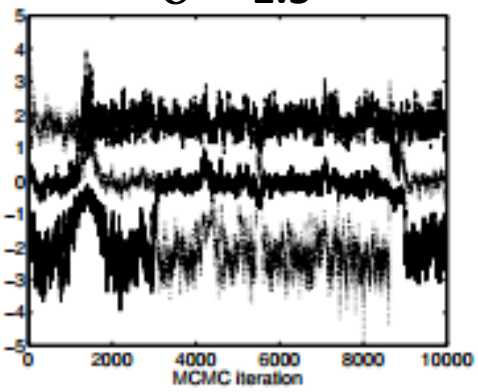
MCMC

σ^2 = 1



MCMC

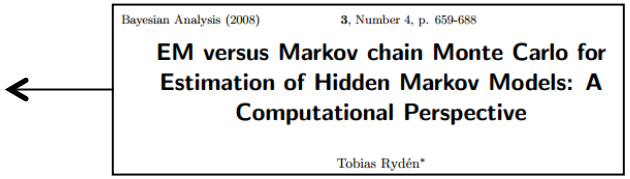
σ^2 = 1.5



Work in Progress

Implementing MCMC by hand in MATLAB. (nontrivial)

Although a Bayesian approach to HMM analysis may be appealing from several perspective, it is the author's experience that users of HMMs often consider writing the computer code necessary to implement such procedures a prohibitive exercise. In particular reversible jump MCMC algorithms have, in the author's view, a somewhat unjustified reputation for being difficult to derive and implement. Still it is clear that readily available software packages would be extremely beneficial for making such methods available to a wider audience of researchers and users in statistics and other scientific fields.



Comparing EM versus MCMC as a function of chain length.

What about chains which don't adequately sample the state space? Does this affect the speed of obtaining λ with MCMC?

At what observational variances does the random 'noise' in MCMC overtake quick convergence to an optimal parameter set?

Are there 'data uncertainty' thresholds which impede progress using MCMC algorithms?

Summary

Markov Chain Monte Carlo (MCMC) algorithms are useful Bayesian inference tools in Hidden Markov Models (HMMs), and can be used to quickly extract an HMM parameter set.

MCMC algorithms can be quicker and less computationally complex than EM algorithms, however their implementation and setup can also be much more complex.

MCMC convergence appears strongly dependent on the amount of data uncertainty. EM shows some dependence on data uncertainty, but continuous sampling always improves inference.

When MCMC does converge on an optimized parameter set λ, it is guaranteed to have a globally maximized likelihood. EM/MLE techniques however may only find parameter sets which are locally maximized.

Thank you for your Attention