# Biostatistics 615/815 Lecture 23: The Baum-Welch Algorithm Advanced Hidden Markov Models

Hyun Min Kang

April 12th, 2011

# Annoucement

## Homework

- Final homework is announced.
- Implementing two among E-M algorithm, Simulated Annealing, and Gibbs Sampler

## 815 Project

- Presentation : Tuesday April 19th.
- Final report : Friday Apil 29th.

## Final Exam

- Thursday April 21st, 10:30AM-12:30PM.

# Key components in 815 Presentations

- Duration : 15 minutes
- Describe / illustrate what the problem is
- Key idea
- Results
- Challenges and lessons from implementations
- Comparisons with other alternatives (if possible)

# Recap - Gibbs Sampler Algorithm

1. Consider a particular choice of parameter values $\lambda^{(t)}$.
2. Define the next set of parameter values by
   - Selecting a component to update, say $i$.
   - Sample value for $\lambda_i^{(t+1)}$, from $p(\lambda_i | x, \lambda_1, \cdots, \lambda_{i-1}, \lambda_{i+1}, \cdots, \lambda_k)$.
3. Increment $t$ and repeat the previous steps.

# Recap - Gibbs Sampling for Gaussian Mixture

- Observed data : $\mathbf{x} = (x_1, \cdots, x_n)$
- Parameters : $\mathbf{z} = (z_1, \cdots, z_n)$ where $z_i \in \{1, \cdots, k\}$.
- Sample each $z_i$ conditioned by all the other $\mathbf{z}$.

# Recap - Simulated Annealing and Gibbs Sampler

## Both Methods are Markov Chains

- The distribution of $\lambda^{(t)}$ only depends on $\lambda^{(t-1)}$
- Update rule defines the transition probabilities between two states, requiring aperiodicity and irreducbility.

## Both Methods are Metropolis-Hastings Algorithms

- Acceptance of proposed update is probabilistically determined by relative probabilities between the original and proposed states

# Today

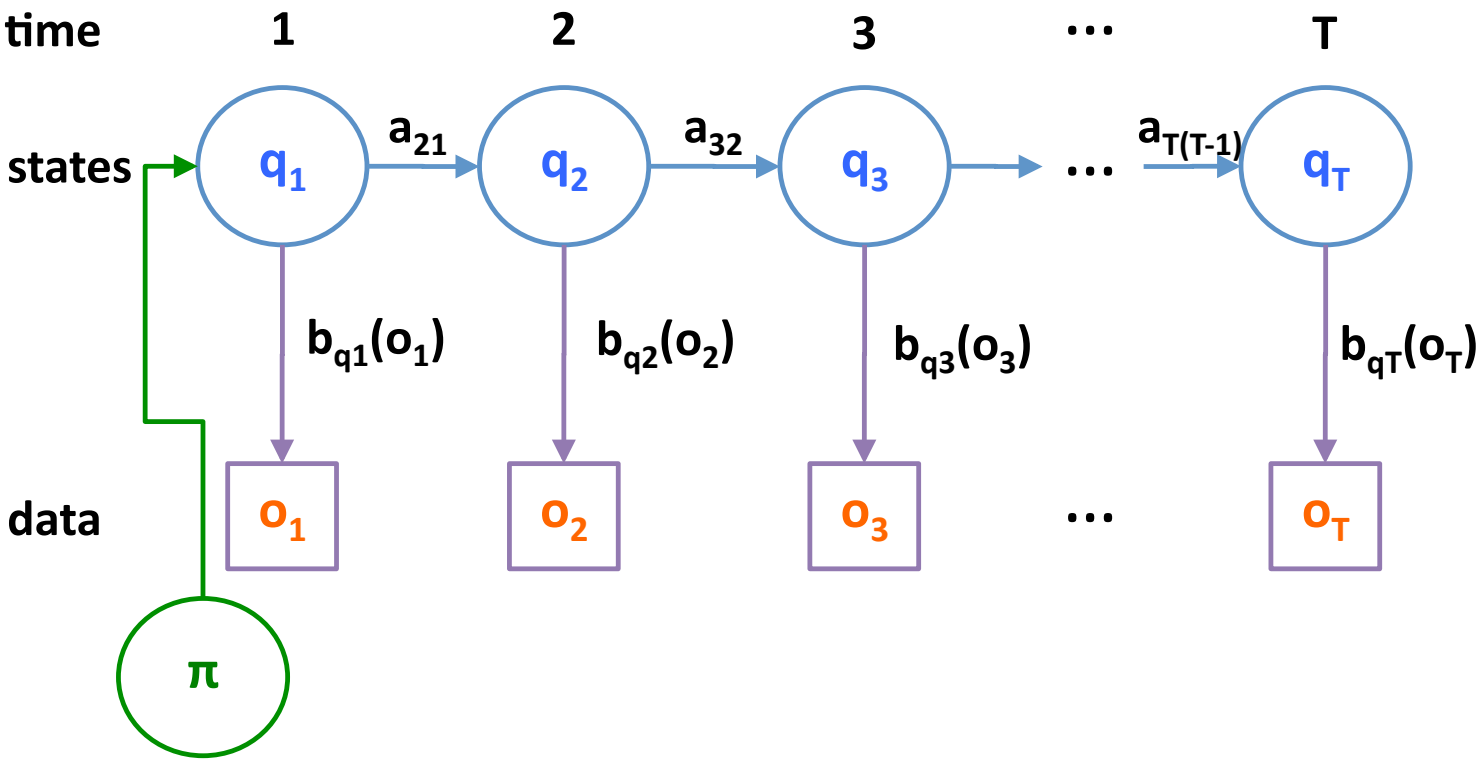
## Baum-Welch Algorithm

- An E-M algorithm for HMM parameter estimation
- Three main HMM algorithms
  - The forward-backward algorithm
  - The Viterbi algorithm
  - The Baum-Welch Algorithm

## Advanced HMM

- Expedited inference with uniform HMM
- Continuous-time Markov Process

# Revisiting Hidden Markov Model



# Statistical analysis with HMM

## HMM for a deterministic problem

- Given
  - Given parameters $\lambda = \{\pi, A, B\}$
  - and data $\mathbf{o} = (o_1, \cdots, o_T)$
- Forward-backward algorithm
  - Compute $\Pr(q_t|\mathbf{o}, \lambda)$
- Viterbi algorithm
  - Compute $\arg\max_{\mathbf{q}} \Pr(\mathbf{q}|\mathbf{o}, \lambda)$

## HMM for a stochastic process / algorithm

- Generate random samples of $\mathbf{o}$ given $\lambda$

# Deterministic Inference using HMM

- If we know the exact set of parameters, the inference is deterministic given data
  - No stochastic process involved in the inference procedure
  - Inference is deterministic just as estimation of sample mean is deterministic
- The computational complexity of the inference procedure is exponential using naive algorithms
- Using dynamic programming, the complexity can be reduced to $O(n^2 T)$.

# Using Stochastic Process for HMM Inference

Using random process for the inference

- Randomly sampling $\mathbf{o}$ from $\Pr(\mathbf{o}|\lambda)$.
- Estimating $\arg\max_\lambda \Pr(\mathbf{o}|\lambda)$.
  - No deterministic algorithm available
  - Simplex, E-M algorithm, or Simulated Annealing is possible apply
- Estimating the distribution $\Pr(\lambda|\mathbf{o})$.
  - Gibbs Sampling

# Recap : The E-M Algorithm

## Expectation step (E-step)

- Given the current estimates of parameters $\theta^{(t)}$, calculate the conditional distribution of latent variable **z**.
- Then the expected log-likelihood of data given the conditional distribution of **z** can be obtained

$$Q(\theta|\theta^{(t)}) = \mathbf{E}_{\mathbf{z}|\mathbf{x},\theta^{(t)}} \left[\log p(\mathbf{x}, \mathbf{z}|\theta)\right]$$

## Maximization step (M-step)

- Find the parameter that maximize the expected log-likelihood

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{t})$$

# Baum-Welch for estimating $\arg\max_\lambda \Pr(\mathbf{o}|\lambda)$

Assumptions

- Transition matrix is identical between states
  - $a_{ij} = \Pr(\mathbf{q}_{t+1} = i|\mathbf{q}_t = j) = \Pr(\mathbf{q}_t = i|\mathbf{q}_{t-1} = j)$
- Emission matrix is identical between states
  - $b_i(j) = \Pr(\mathbf{o}_t = j|\mathbf{q}_t = i) = \Pr(\mathbf{o}_{t=1} = j|\mathbf{q}_{t-1} = i)$
- This is NOT the only possible assumption.
  - For example, $a_{ij}$ can be parameterized as a function of $t$.
  - Multiple sets of $\mathbf{o}$ independently drawn from the same distribution can be provided.
  - Other assumptions will result in different formulation of E-M algorithm

# E-step of the Baum-Welch Algorithm

1. Run the forward-backward algorithm given $\lambda^{(\tau)}$

$$\begin{aligned}
\alpha_t(i) &= \Pr(o_1, \cdots, o_t, q_t = i|\lambda^{(\tau)}) \\
\beta_t(i) &= \Pr(o_{t+1}, \cdots, o_T|q_t = i, \lambda^{(\tau)}) \\
\gamma_t(i) &= \Pr(q_t = i|\mathbf{o}, \lambda^{(\tau)}) = \frac{\alpha_t(i)\beta_t(i)}{\sum_k \alpha_t(k)\beta_t(k)}
\end{aligned}$$

2. Compute $\xi_t(i,j)$ using $\alpha_t(i)$ and $\beta_t(i)$

$$\begin{aligned}
\xi_t(i,j) &= \Pr(q_t = i, q_{t+1} = j|\mathbf{o}, \lambda^{(\tau)}) \\
&= \frac{\alpha_t(i)a_{ji}b_j(o_{t+1})\beta_{t+1}(j)}{\Pr(\mathbf{o}|\lambda^{(\tau)})} \\
&= \frac{\alpha_t(i)a_{ji}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{(k,l)} \alpha_t(k)a_{lk}b_l(o_{t+1})\beta_{t+1}(l)}
\end{aligned}$$

# M-step of the Baum-Welch Algorithm

Let $\lambda^{(\tau+1)} = (\pi^{(\tau+1)}, A^{(\tau+1)}, B^{(\tau+1)})$

$$\begin{aligned}
\pi^{(\tau+1)}(i) &= \frac{\sum_{t=1}^{T} \Pr(q_t = i|\mathbf{o}, \lambda^{(\tau)})}{T} = \frac{\sum_{t=1}^{T} \gamma_t(i)}{T} \\
a_{ij}^{(\tau+1)} &= \frac{\sum_{t=1}^{T-1} \Pr(q_t = j, q_{t+1} = i|\mathbf{o}, \lambda^{(\tau)})}{\sum_{t=1}^{T-1} \Pr(q_t = j|\mathbf{o}, \lambda^{(\tau)})} = \frac{\sum_{t=1}^{T-1} \xi_t(j,i)}{\sum_{t=1}^{T-1} \gamma_t(j)} \\
b_i(k)^{(\tau+1)} &= \frac{\sum_{t=1}^{T} \Pr(q_t = i, o_t = k|\mathbf{o}, \lambda^{(\tau)})}{\sum_{t=1}^{T} \Pr(q_t = i|\mathbf{o}, \lambda^{(\tau)})} = \frac{\sum_{t=1}^{T} \gamma_t(i) I(o_t = k)}{\sum_{t=1}^{T} \gamma_t(i)}
\end{aligned}$$

A detailed derivation can be found at

- Welch, "Hidden Markov Models and The Baum Welch Algorithm", IEEE Information Theory Society News Letter, Dec 2003

# Implementing HMM Algorithms

```
class HMM615 {
public:
  int T;                        // number of instances
  int N;                        // number of states
  int O;                        // number of possible values
  std::vector<double> pis;      // pis[i]       : Pr(q_0=i)
  std::vector<double> trans;    // trans[i*N+j] : Pr(q_t=j|q_{t-1}=i)
  double symmTrans;
  std::vector<double> emis;     // emis[i*N+j]  : Pr(o_t=j|q_t=i)
  std::vector<int> obs;         // obs[t] : observed data from 0..O

  std::vector<double> alphas; // alphas[t*n+i] : Pr(o_{1..t},q_t=i|lambda)
  std::vector<double> betas;  // betas[t*n+i]  : Pr(o_{t+1..T}|q_t=i,lambda)
  std::vector<double> gammas; // gammas[t*n+i] : Pr(q_t=i|lambda,o_{1..T})

  double computeForwardBackward();
  double computeBaumWelch(double tol);
};
```

# Implementing Forward algorithm : $\alpha_t(i)$

```
void HMM615::computeForwardBackward() {
  double sum = 0;
  // initialize alpha values
  for(int i=0; i < N; ++i)
    sum += (alphas[0*N + i] = pis[i] * emis[i*O+obs[0]]);
  for(int i=0; i < N; ++i) alphas[0*N + i] /= sum;  // normalize sum(alphas)
  // iterate over alphas for each t
  for(int t=1; t < T; ++t) {
    for(int i=0; i < N; ++i) {
      alphas[t*N + i] = sum = 0;
      for(int j=0; j < N; ++j)
        alphas[t*N + i] += (alphas[(t-1)*N +j] * trans[j*N + i]);
      sum += (alphas[t*N + i] *= emis[i*O+obs[t]]);
    }
    for(int i=0; i < N; ++i) alphas[t*N + i] /= sum;  // normalize sum(alphas)
  }
  // (continued to next slides)
```

# Implementing Backward algorithm : $\beta_t(i)$

```
// initialize the last element of betas
sum = 0;
for(int i=0; i < N; ++i) sum += emis[i*O+obs[T-1]];
for(int i=0; i < N; ++i) betas[(T-1)*N + i] = 1./sum;
// main body of backward algorithm
for(int t=T-2; t >=0; --t) {
  for(int i=0; i < N; ++i) {
    betas[t*N + i] = sum = 0;
    for(int j=0; j < N; ++j)
      betas[t*N + i] += (betas[(t+1)*N + j] * trans[i*N + j]
                         * emis[j*O+obs[t+1]]);
    sum += (betas[t*N+i] * emis[i*O+obs[t]]);
  }
  sum = 0;
  // normalize sum (betas * emis )
  for(int i=0; i < N; ++i) betas[t*N + i] /= sum;
}
// (continued to next slide)
```

# Combining forward-backward algorithm : $\gamma_t(i)$

```
  // compute gammas = Pr(q_t|o,lambda)
  for(int t=0; t < T; ++t) {
    sum = 0;
    for(int i=0; i < N; ++i)
      sum += (gammas[t*N + i] = alphas[t*N + i] * betas[t*N + i]);

    for(int i=0; i < N; ++i)
      gammas[t*N + i] /= sum;  // normalize sum(gammas)
  }
}
```

# Implementation Notes : Forward-backward algorithm

- Forward-backward algorithm for arbitrary numbers of states and observations.
- Normalization of $\sum_t \alpha_t(i)$ and $\sum_i \beta_t(i) b_i(o_t)$ at each step
  - Only relative values of $\alpha_t(i)$ and $\beta_t(i)$ are important
  - Avoids potential overflow / underflow due to numerical precision
  - Why normalize $\sum_i \beta_t(i) b_i(o_t)$ instead of $\sum_i \beta_t(i)$?

# A small utility function : update

```
// assign newVal to dst, after computing the relative differences between them
// note that dst is call-by-reference, and newVal is call-by-value
double HMM615::update(double& dst, double newVal) {
   // calculate the relative differences
   double relDiff = fabs((dst-newVal)/(newVal+ZEPS));
   // update the destination value
   dst = newVal;
   // update the destination value
   return relDiff;
}
```

# The Baum-Welch Algorithm : Initialization

```
void HMM615::computeBaumWelch(double tol) {
  double tmp, sum, relDiff = 1e9;

  std::vector<double> sumGammas(N,0), sumObsGammas(N*O,0);
  std::vector<double> xis(N*N,0), sumXis(N*N,0);

  // iterate until the difference is small enough
  for(int iter = 0; (iter < MAX_ITERATION) && ( relDiff > tol ); ++iter) {
    relDiff = 0;
    computeForwardBackward(); // run forward-backward algorithm
    for(int i=0; i < N; ++i)   sumGammas[i] = 0;
    for(int i=0; i < N*O; ++i) sumObsGammas[i] = 0;
    for(int i=0; i < N*N; ++i) xis[i] = sumXis[i] = 0;
```

# The Baum-Welch Algorithm : E-step

```
// calculate \sum_t gamma_t(i)
for(int t=0; t < T; ++t) {
  for(int i=0; i < N; ++i) {
    sumGammas[i] += gammas[t*N + i];
    sumObsGammas[i*O + obs[t]] += gammas[t*N + i];
  }
}
// calcupate \sum_t xi_t(i,j)
for(int t=0; t < T-1; ++t) {
   sum = 0;
   for(int i=0; i < N; ++i) {
     for(int j=0; j < N; ++j)
       sum += ( xis[i*N+j] = ( alphas[t*N + i] * trans[i*N + j] *
                              betas[(t+1)*N + j] *emis[j*O + obs[t+1] ] ) );
   }
   for(int i=0; i < N*N; ++i)  sumXis[i] += (xis[i] /= sum);
}
```

# The Baum-Welch Algorithm : M-step

```
    // update transition matrix
    for(int i=0; i < N; ++i) {
      for(int j=0; j < N; ++j) {
         tmp = sumXis[i*N+j] / (sumGammas[i]-gammas[(T-1)*N+i]+ZEPS);
         relDiff += update( trans[i*N+j], tmp );
      }
    }

    // update pis and emission matrix
    for(int i=0; i < N; ++i) {
      relDiff += update( pis[i], sumGammas[i]/T );
      for(int j=0; j < O; ++j)
        relDiff += update (emis[i*O+j], sumObsGammas[i*O+j]/(sumGammas[i]+ZEPS));
    }
  }  // repeat until relative difference is small enough
}
```

# A working example : Biased coin example

## Model

- Observations : $O = \{1(Head), 2(Tail)\}$
- Hidden states : $S = \{1(Fair), 2(Biased)\}$
- Initial states : $\pi = \{0.8, 0.2\} = A^{\infty}\pi_0$
- Transition probability : $A(i,j) = a_{ij} = \begin{pmatrix} 0.95 & 0.2 \\ 0.05 & 0.8 \end{pmatrix}$
- Emission probability : $B(i,j) = b_j(i) = \begin{pmatrix} 0.5 & 0.9 \\ 0.5 & 0.1 \end{pmatrix}$

# Biased coin example : main() function

```
int main(int argc, char** argv) {
  // input/output routines ... omitted
  // assign an initial starting point
  double theta0 = 0.1;
  std::vector<double> trans(4, theta0);
  std::vector<double> emis(4, 0.5);
  std::vector<double> pis(2, 0.5);  // initial pi = (0.5, 0.5)
  trans[0] = trans[3] = 1-theta0;   // initial A = (0.9, 0.1; 0.1, 0.9)
  emis[2] = 0.7; emis[3] = 0.3;     // initial B = (0.5, 0.7; 0.5; 0.3)

  HMM615 hmm(pis, trans, emis, observations); // constructor omitted in the slides
  hmm.computeBaumWelch(1e-3);       // run Baum-Welch algorithm

  return 0;
}
```

# Biased coin example : Results

```
user@host:~/ > baumWelchCoin biasedCoinInput.10000.txt
Iteration 216,
normDiff = 0.000982141,
pis = (0.80249, 0.19751),
trans = (0.942284, 0.0577156, 0.234439, 0.765561),
emis = (0.493703, 0.506297, 0.905311, 0.0946887)

user@host:~/ > baumWelchCoin biasedCoinInput.1000.txt
Iteration 621,
normDiff = 0.00099904,
pis = (0.544055, 0.455945),
trans = (0.723604, 0.276396, 0.330778, 0.669222),
emis = (0.291926, 0.708074, 0.904004, 0.0959957)
```

# Summary : Baum-Welch Algorithm

- E-M algorithm for estimating HMM parameters
- Assumes identical transition and emission probabilities across $t$
- The framework can be accomodated for differently contrained HMM
- Requires many observations to reach a reliable estimates

# Rapid Inference with Uniform HMM

## Uniform HMM

- Definition
  - $\pi_i = 1/n$
  - $a_{ij} = \begin{cases} \theta & i \neq j \\ 1-(n-1)\theta & i = j \end{cases}$
  - $b_i(k)$ has no restriction.
- Independent transition between $n$ states
- Useful model in genetics and speech recognition.

# Rapid Inference with Uniform HMM

## Uniform HMM

- Definition
  - $\pi_i = 1/n$
  - $a_{ij} = \begin{cases} \theta & i \neq j \\ 1-(n-1)\theta & i = j \end{cases}$
  - $b_i(k)$ has no restriction.
- Independent transition between $n$ states
- Useful model in genetics and speech recognition.

## The Problem

- The time complexity of HMM inference is $O(n^2 T)$.
- For large $n$, this still can be a substantial computational burden.
- Can we reduce the time complexity by leveraging the simplicity?

# Forward Algorithm with Uniform HMM

## Original Forward Algorithm

$$\alpha_t(i) = \Pr(o_1, \cdots, o_t, q_t = i|\lambda) = \left[\sum_{j=1}^{n} \alpha_{t-1}(j) a_{ij}\right] b_i(o_t)$$

## Rapid Forward Algorithm for Uniform HMM

$$\begin{aligned}
\alpha_t(i) &= \left[\sum_{j=1}^{n} \alpha_{t-1}(j) a_{ij}\right] b_i(o_t) \\
&= \left[(1-(n-1)\theta)\alpha_{t-1}(i) + \sum_{j\neq i} \alpha_{t-1}(j)\theta\right] b_i(o_t) \\
&= \left[(1-n\theta)\alpha_{t-1}(i) + \theta\right] b_i(o_t)
\end{aligned}$$

- Assuming normalizaed $\sum_i \alpha_t(i) = 1$ for every $t$.
- The total time complexity is $O(nT)$.

# Backward Algorithm with Uniform HMM

## Original Forward Algorithm

$$\beta_t(i) = \Pr(o_{t+1}, \cdots, o_T | q_t = i, \lambda) = \sum_{j=1}^{n} \beta_{t+1}(j) a_{ji} b_j(o_{t+1})$$

## Rapid Forward Algorithm for Uniform HMM

$$\begin{aligned}
\beta_t(i) &= \sum_{j=1}^{n} \beta_{t+1}(j) a_{ji} b_j(o_{t+1}) \\
&= (1-(n-1)\theta)\beta_{t+1}(i) b_i(o_{t+1}) + \theta \textstyle\sum_{j \neq i} \beta_{t+1}(j) b_j(o_{t+1}) \\
&= (1-n\theta)\beta_{t+1}(i) b_t(o_{t+1}) + \theta
\end{aligned}$$

Assuming $\sum_i \beta_t(i) b_i(o_t) = 1$ for every $t$. (Now we know why!)

# Summary : Uniform HMM

- Rapid computation of forward-backward algorithm leveraging symmetric structure
- Rapid Baum-Welch algorithm is also possible in a similar manner
- It is important to understand the computational details of exisitng methods to further tweak the method when necessary.

# Continuous-time Markov Process (CTMP)

## Example : Single dimensional Brownian motion

- A particle is moving along a line at a constant velocity $v$.
- At a rate $\lambda$ times per second, the particle changes the direction.
- The position of particle is observed at arbitrary time points $t_1, t_2, \cdots, t_n$.
- How can we model the trajectory of the particles given observations?

## Other Applications

- Queueing theory
- Modeling recombination in diploid organisms
- Many other infinitesimal Models

# Key Idea of CTMP

## Transition Rate instead of Transition Probability

- In discret MP, $a_{ij}$ defines the transition probability between states
- In CTMP, transition rate $\rho$ defines the transition probability within some time intervals.

$$\Pr(q_s = i, \quad \forall s \in (t, t+r) | q_t = i) = \exp(-\rho_{ii} r)$$

## Difference from discret HMM

- The transition probability between time points are no longer identical
- However, the transition probability can be parameterized using $\rho$ and the interval size
- Developing E-M algorithm for estimating $\rho$ is more sophisticated than the Baum-Welch algorithm

# Summary

## Today

- The Baum-Welch Algorithm
- Rapid inference with Uniform HMM
- Continuous-time Markov Process (CTMP)

## Next Lecture

- Overview for the Final Exam