

Biostatistics 615/815 Lecture 20: The Baum-Welch Algorithm Advanced Hidden Markov Models

Hyun Min Kang

November 22nd, 2011

Today

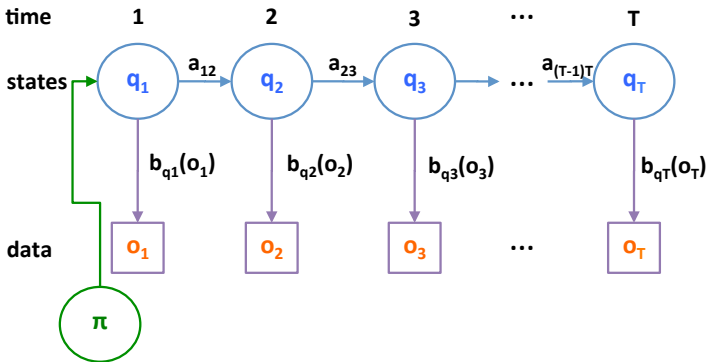
Baum-Welch Algorithm

- 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 $\mathbf{o} = (o_1, \dots, o_T)$
- 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 λ

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 analytic 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 \mathbf{z} .
- Then the expected log-likelihood of data given the conditional distribution of \mathbf{z} can be obtained

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

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
 - $a_{ij} = \Pr(\mathbf{q}_{t+1} = j | \mathbf{q}_t = i) = \Pr(\mathbf{q}_t = j | \mathbf{q}_{t-1} = i)$
- 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 configurations of HMM
 - 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 $\lambda^{(\tau)}$

$$\alpha_t(i) = \Pr(o_1, \dots, o_t, q_t = i | \lambda^{(\tau)})$$

$$\beta_t(i) = \Pr(o_{t+1}, \dots, o_T | q_t = i, \lambda^{(\tau)})$$

$$\gamma_t(i) = \Pr(q_t = i | \mathbf{o}, \lambda^{(\tau)}) = \frac{\alpha_t(i)\beta_t(i)}{\sum_k \alpha_t(k)\beta_t(k)}$$

E-step of the Baum-Welch Algorithm

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

$$\alpha_t(i) = \Pr(o_1, \dots, o_t, q_t = i | \lambda^{(\tau)})$$

$$\beta_t(i) = \Pr(o_{t+1}, \dots, o_T | q_t = i, \lambda^{(\tau)})$$

$$\gamma_t(i) = \Pr(q_t = i | \mathbf{o}, \lambda^{(\tau)}) = \frac{\alpha_t(i)\beta_t(i)}{\sum_k \alpha_t(k)\beta_t(k)}$$

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

$$\xi_t(i, j) = \Pr(q_t = i, q_{t+1} = j | \mathbf{o}, \lambda^{(\tau)})$$

$$= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\Pr(\mathbf{o} | \lambda^{(\tau)})}$$

$$= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{(k,l)} \alpha_t(k) a_{kl} b_l(o_{t+1}) \beta_{t+1}(l)}$$

E-step : $\xi_t(i, j)$

$$\xi_t(i, j) = \Pr(q_t = i, q_{t+1} = j | \mathbf{o}, \lambda^{(\tau)})$$

- Quantifies joint state probability between consecutive states
- Need to estimate transition probability
- Requires $O(n^2 T)$ memory to store entirely.
 - Only $O(n^2)$ is necessary for running Baum-Welch algorithm

M-step of the Baum-Welch Algorithm

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

$$\pi^{(\tau+1)}(i) = \frac{\sum_{t=1}^T \Pr(q_t = i | \mathbf{o}, \lambda^{(\tau)})}{T} = \frac{\sum_{t=1}^T \gamma_t(i)}{T}$$

$$a_{ij}^{(\tau+1)} = \frac{\sum_{t=1}^{T-1} \Pr(q_t = i, q_{t+1} = j | \mathbf{o}, \lambda^{(\tau)})}{\sum_{t=1}^{T-1} \Pr(q_t = i | \mathbf{o}, \lambda^{(\tau)})} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_i(k)^{(\tau+1)} = \frac{\sum_{t=1}^T \Pr(q_t = i, o_t = k | \mathbf{o}, \lambda^{(\tau)})}{\sum_{t=1}^T \Pr(q_t = i | \mathbf{o}, \lambda^{(\tau)})} = \frac{\sum_{t=1}^T \gamma_t(i) I(o_t = k)}{\sum_{t=1}^T \gamma_t(i)}$$

A detailed derivation can be found at

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

Additional function to HMM615.h

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

Handling large number of states

```
class HMM615 {
...
void normalize(std::vector<double>& v) { // additional function
    double sum = 0;
    for(int i=0; i < (int)v.size(); ++i) sum += v[i];
    for(int i=0; i < (int)v.size(); ++i) v[i] /= sum;
}
void forward() {
    for(int i=0; i < nStates; ++i)
        alphas.data[0][i] = pis[i] * emis.data[i][outs[0]];
    for(int t=1; t < nTimes; ++t) {
        for(int i=0; i < nStates; ++i) {
            alphas.data[t][i] = 0;
            for(int j=0; j < nStates; ++j) {
                alphas.data[t][i] += (alphas.data[t-1][j] * trans.data[j][i]
                    * emis.data[i][outs[t]]);
            }
        }
        normalize(alphas.data[t]); // **ADD THIS LINE**
    }
}
...
};
```

Additional function to Matrix615.h

```
void Matrix615::fill(T val) {
    int nr = rowNums();
    for(int i=0; i < nr; ++i) {
        std::fill(data[i].begin(),data[i].end(),val);
    }
}

// print the content of matrix
void Matrix615::print(std::ostream& o) {
    int nr = rowNums();
    int nc = colNums();

    for(int i=0; i < nr; ++i) {
        for(int j=0; j < nc; ++j) {
            if ( j > 0 ) o << "\t";
            o << data[i][j];
        }
        o << std::endl;
    }
}
```

Baum-Welch algorithm : initialization

```
// return a pair of (# iter, relative diff) given tolerance
std::pair<int,double> HMM615::baumWelch(double tol) {
    // temporary variables to use internally

    Matrix615<double> xis(nStates,nStates); // Pr(q_{t+1} = j | q_t = j)
    Matrix615<double> sumXis(nStates,nStates); // sum_t xis(i,j)
    Matrix615<double> sumObsGammas(nStates,nObs); // sum_t gammas(i)I(o_t=j)
    std::vector<double> sumGammas(nStates); // sum_t gammas(i)
    double tmp, sum, relDiff = 1.;

    int iter;
    for(iter=0; (iter < MAX_ITERATION) && ( relDiff > tol ); ++iter) {
        relDiff = 0;

        // E-step : compute Pr(q|o,lambda)
        forwardBackward();
```


Baum-Welch algorithm : M-step

```
// initialize temporary storage
std::fill(sumGammas.begin(),sumGammas.end(),0);
sumXis.fill(0); sumObsGammas.fill(0); xis.fill(0);

// M-step : updates pis, trans, and emis
for(int t=0; t < nTimes-2; ++t) {
    sum = 0; // sum stores sum of xis
    for(int i=0; i < nStates; ++i)
        for(int j=0; j < nStates; ++j)
            sum += (xis.data[i][j] = alphas.data[t][i] * trans.data[i][j]
                    * betas.data[t+1][j] * emis.data[j][outs[t+1]]);
    // update sumGammas, sumObsGammas, sumXis
    for(int i=0; i < nStates; ++i) {
        sumGammas[i] += gammas.data[t][i];
        sumObsGammas.data[i][outs[t]] += gammas.data[t][i];
        for(int j=0; j < nStates; ++j)
            sumXis.data[i][j] += (xis.data[i][j] /= sum);
    }
}
```

Baum-Welch algorithm : M-step

```
for(int i=0; i < nStates; ++i) {
    relDiff += update( pis[i], sumGammas[i]/(nTimes-1) );
    for(int j=0; j < nStates; ++j) {
        relDiff += update(trans.data[i][j],
            sumXis.data[i][j] / (sumGammas[i] - gammas.data[nTimes-1][i] + ZEPS));
    }
    for(int j=0; j < nObs; ++j ) {
        relDiff += update(emis.data[i][j],
            sumObsGammas.data[i][j] / (sumGammas[i] + ZEPS) );
    }
}
}
return std::pair<int,double>(iter,relDiff);
}
```

A working example : Biased dice example

- Observations : $O = \{1, 2, \dots, 6\}$
- Hidden states : $S = \{FAIR, 1 - BIASED, \dots, 6 - BIASED\}$
- Priors : $\pi = \{0.70, 0.05, 0.05, \dots, 0.05\}$
- Transition matrix :

$$A = \begin{pmatrix} 0.94 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.94 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.94 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.94 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.94 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.94 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.94 \end{pmatrix}$$

- Emission matrix :

$$B = \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0.95 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.95 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.95 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.95 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.95 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.95 \end{pmatrix}$$

Biased dice example : main() function

```

#include <iostream>
#include <iomanip>
#include "Matrix615.h"
#include "HMM615.h"
int main(int argc, char** argv) {
    if ( argc != 5 ) {
        std::cerr << "Usage: baumWelch [trans0] [emis0] [pis0] [obs]" << std::endl;
        return -1;
    }

    std::vector<int> obs;
    Matrix615<double> trans(argv[1]);
    int ns = trans.rowNums();
    if ( ns != trans.colNums() ) {
        std::cerr << "Transition matrix is not square" << std::endl;
        return -1;
    }

    Matrix615<double> emis(argv[2]);

```

Biased dice example : main() function (cont'd)

```
if ( ns != emis.rowNums() ) {
    std::cerr << "Emission and transition matrices do not match" << std::endl;
    return -1;
}
int no = emis.colNums();

readFromFile<int> (obs, argv[4]);
int nt = (int)obs.size();

HMM615 hmm(ns, no, nt);

readFromFile<double> (hmm.pis, argv[3]);
if ( ns != (int)hmm.pis.size() ) {
    std::cerr << "Transition and Prior matrices do not match" << std::endl;
    return -1;
}
hmm.trans = trans;
hmm.emis = emis;
hmm.outs = obs;
```

Biased dice example : main() function (cont'd)

```
std::pair<int,double> result = hmm.baumWelch(1e-6);

std::cout << "# ITERATIONS : " << result.first << "\t";
std::cout << "SUM RELATIVE DIFF : " << result.second << std::endl;
std::cout << std::fixed << std::setprecision(5);
std::cout << "PIS:" << std::endl;
for(int i=0; i < ns; ++i) {
    if ( i > 0 ) std::cout << "\t";
    std::cout << hmm.pis[i];
}
std::cout << std::endl;
std::cout << "-----" << std::endl;
std::cout << "TRANS:" << std::endl;
hmm.trans.print(std::cout);
std::cout << "-----" << std::endl;
std::cout << "EMIS:" << std::endl;
hmm.emis.print(std::cout);
std::cout << "-----" << std::endl;
return 0;
}
```

Biased dice example : Results with 20,000 samples

```
$ ./baumWelch trans.dice.txt emis.dice.txt pis.dice.txt outs.dice.txt
```

```
# ITERATIONS : 23          SUM RELATIVE DIFF : 9.80431e-07
```

```
PIS:
```

```
0.14951 0.13582 0.12723 0.16745 0.14518 0.13736 0.13739
```

```
-----
```

```
TRANS:
```

```
0.94159 0.00924 0.01111 0.01126 0.00741 0.01374 0.00569
```

```
0.01018 0.93318 0.00870 0.01134 0.00943 0.01317 0.01401
```

```
0.00604 0.01632 0.93780 0.00884 0.01215 0.01029 0.00855
```

```
0.01189 0.00874 0.00855 0.94798 0.00932 0.00724 0.00641
```

```
0.01172 0.00859 0.00735 0.00957 0.94645 0.00688 0.00944
```

```
0.00874 0.00969 0.01066 0.01206 0.00886 0.93777 0.01235
```

```
0.01215 0.01143 0.00803 0.01010 0.00774 0.00876 0.94180
```

```
-----
```

```
EMIS:
```

```
0.16357 0.16859 0.17141 0.16421 0.15995 0.17227
```

```
0.94908 0.01052 0.01026 0.01450 0.00669 0.00895
```

```
0.00624 0.95081 0.01055 0.01392 0.00936 0.00912
```

```
0.01113 0.01117 0.95023 0.00826 0.00948 0.00974
```

```
0.01012 0.00871 0.00989 0.95121 0.01051 0.00956
```

```
0.00877 0.00848 0.00983 0.00827 0.95513 0.00951
```

```
0.00956 0.00874 0.00722 0.00981 0.01338 0.95128
```

Biased dice example : Starting with uniform parameters

```
$ ./baumWelch trans0.dice.txt emis0.dice.txt pis0.dice.txt outs.dice.txt
```

```
# ITERATIONS : 2          SUM RELATIVE DIFF : 0
```

```
PIS:
```

```
0.14285 0.14285 0.14285 0.14285 0.14285 0.14285 0.14285
```

```
-----
```

```
TRANS:
```

```
0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286
```

```
0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286
```

```
0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286
```

```
0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286
```

```
0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286
```

```
0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286
```

```
0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286
```

```
-----
```

```
EMIS:
```

```
0.16002 0.15312 0.19127 0.17027 0.16217 0.16317
```

```
0.16002 0.15312 0.19127 0.17027 0.16217 0.16317
```

```
0.16002 0.15312 0.19127 0.17027 0.16217 0.16317
```

```
0.16002 0.15312 0.19127 0.17027 0.16217 0.16317
```

```
0.16002 0.15312 0.19127 0.17027 0.16217 0.16317
```

```
0.16002 0.15312 0.19127 0.17027 0.16217 0.16317
```

```
0.16002 0.15312 0.19127 0.17027 0.16217 0.16317
```


Starting with incorrect emission matrix

```
$ cat emis1.dice.txt
0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667
0.5 0.1 0.1 0.1 0.1 0.1
0.1 0.5 0.1 0.1 0.1 0.1
0.1 0.1 0.5 0.1 0.1 0.1
0.1 0.1 0.1 0.5 0.1 0.1
0.1 0.1 0.1 0.1 0.5 0.1
0.1 0.1 0.1 0.1 0.1 0.5
```

Starting with incorrect emission matrix

```
$ ./baumWelch trans0.dice.txt emis1.dice.txt pis0.dice.txt outs.dice.txt
```

```
# ITERATIONS : 37          SUM RELATIVE DIFF : 7.50798e-07
```

```
PIS:
```

```
0.14951 0.13582 0.12723 0.16745 0.14518 0.13736 0.13739
```

```
-----
```

```
TRANS:
```

```
0.94159 0.00924 0.01111 0.01126 0.00741 0.01374 0.00569
```

```
0.01018 0.93318 0.00870 0.01134 0.00943 0.01317 0.01401
```

```
0.00604 0.01632 0.93780 0.00884 0.01215 0.01029 0.00855
```

```
0.01189 0.00874 0.00855 0.94798 0.00932 0.00724 0.00641
```

```
0.01172 0.00859 0.00735 0.00957 0.94645 0.00688 0.00944
```

```
0.00874 0.00969 0.01066 0.01206 0.00886 0.93777 0.01235
```

```
0.01215 0.01143 0.00803 0.01010 0.00774 0.00876 0.94180
```

```
-----
```

```
EMIS:
```

```
0.16357 0.16859 0.17141 0.16421 0.15995 0.17227
```

```
0.94908 0.01052 0.01026 0.01450 0.00669 0.00895
```

```
0.00624 0.95081 0.01055 0.01392 0.00936 0.00912
```

```
0.01113 0.01117 0.95023 0.00826 0.00948 0.00974
```

```
0.01012 0.00871 0.00989 0.95121 0.01051 0.00956
```

```
0.00877 0.00848 0.00983 0.00827 0.95513 0.00951
```

```
0.00956 0.00874 0.00722 0.00981 0.01338 0.95128
```

Summary : Baum-Welch Algorithm

- E-M algorithm for estimating HMM parameters
- Assumes identical transition and emission probabilities across t
- The framework can be accommodated for differently constrained 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

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

Rapid Forward Algorithm for Uniform HMM

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

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

Backward Algorithm with Uniform HMM

Original Forward Algorithm

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

Rapid Forward Algorithm for Uniform HMM

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

Assuming $\sum_i \beta_t(i) b_i(o_t) = 1$ for every t .

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 existing methods to further tweak the method when necessary.

Summary

Today

- The Baum-Welch Algorithm
- Rapid inference with Uniform HMM

Next Lecture

- Linear Algebra in C++