COMP 790-124, HW3

Wile E. Coyote

November 10, 2012

Deadline: 11/9/2012 11:59PM EST
Submit `hw3.pdf` by e-mail, `mailto:vjojic+comp790+hw3@cs.unc.edu`.

We will assume existence of an N long backbone sequence s in an N. In this assignment the alphabet will be of size 4, corresponding to nucleotides. We will construct a hidden Markov model that generates a shorter sequence from the backbone sequence. The shorter sequence will consist of two parts of equal length L. First part of sequence corresponds to offsets, 1 through L, and the second part of the sequence corresponds to offsets, $L+1$ to $2L$. With each offset i in the two-part sequence we will associate a hidden variable that points to a position in the backbone sequence. The probability of a letter x_i, given pointer h_i is

$$p(x_i|h_i) = \begin{cases} 0.99, & \text{if } x_i = s_{h_i} \\ \frac{0.01}{a-1}, & \text{if } x_i \neq s_{h_i}. \end{cases}$$

Finally, we define transition probability on the h_i

$$h_{i+1}|h_i \propto \begin{cases} \text{TruncPoiss}(h_{i+1} - h_i), & \text{if } i = L \\ \pi_{\text{ins}}, & \text{if } h_{i+1} = h_i, i \neq L \\ \pi_{\text{del}}, & \text{if } h_{i+1} = h_i + 2, i \neq L, h_i + 2 \leq N \\ \pi_{\text{copy}}, & \text{if } h_{i+1} = h_i + 1, i \neq L, h_i + 1 \leq N. \end{cases}$$

where TruncPoiss denotes a truncated poisson, given by

$$\text{TruncPoiss}(l) \propto \begin{cases} \text{Poiss}(l, \lambda = 100), & \text{if } 90 \leq l \leq 110 \\ 0, & \text{otherwise} \end{cases}$$

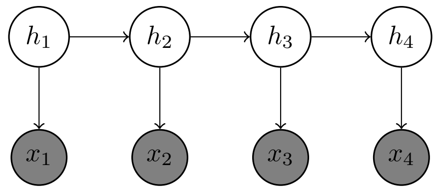
Hence, we move the pointer forward with an occasional skip (deletion) or lag (insert), except when we reach offset L where we make a leap of approximately 100 positions.

We will use a `logsum` function

```
function s = logsum(vec)
m = max(vec);
s = log(sum(exp(vec - m))) + m;
```

and `logProbPoiss`

```
function logProb = logProbTruncPoiss(i,lambda)
logProb = log(lambda)*i + (-lambda) - sum(log(1:i));
```



Problem 1(3pt) Given the HMM model specification above implement a forward pass in the HMM. Storing the transition matrix explicitly would be too costly. But the matrix is very sparse. With the exception of $i = L$, from a given offset h_i we can transition to *at most* three different offsets. Hence computation of the forward pass will require us to explicitly account for these possibilities. Note that if $h_i = N$ then the only possible transition is to $h_{i+1} = N$.

Note that transition from h_L to h_{L+1} is made according to a Poisson.

```
function m_f = fw(s,x,pins,pdel,pcopy)
N = length(s);
L = length(x)/2;
m_f = -realmax*ones(N,2*L);

logpins = log(pins);
logpdel = log(pdel);
logpcopy = log(pcopy);
for a=1:4
    for b=1:4
        logmut(a,b) = log(0.99)*(a==b) + log(0.01)*(a~=b);
    end
end

m_f(1:N,1) = -log(N) + logmut(s,x(1));
for i=2:2*L
    for prev=1:N
        if i~=L+1
            % insert
            vala = m_f(prev,i);
            valb = m_f(prev,i-1)+ logmut(s(prev),x(i)) + logpins;
            m_f(prev,i) = logsum([vala valb]);
            if prev<=N-2
                % delete
                vala = m_f(prev+2,i);
                valb  =  m_f(prev+2,i-1) + logmut(s(prev+2), x(i)) + logpdel;
```

```
                m_f(prev+2,i) = logsum([vala valb]);
            end
            if prev<=N-1
                % copy
                vala = m_f(prev+1,i);
                valb  =  m_f(prev+1,i-1) + logmut(s(prev+1), x(i)) + logpcopy;
                m_f(prev+1,i) = logsum([vala valb]);
            end
        else
            if prev+90<=N-1
                for next=prev+90:min(prev+110,N-1)
                    vala = m_f(next,i);
                    valb  =  logProbTruncPoiss(m_f(next+1, i) - m_f(next, i), 100);
                    m_f(next,i) = logsum([vala valb]);
                end
            end
        end
    end
end
```

Run forward pass on inputs stored in `hw3.mat` and run this script

```
m_f = fw(seq,x,0.005,0.005,0.99);
logProb = logsum(m_f(:,end))
```

The resulting `logProb` is -141.1672

Problem 2(3pt) Implement a backward pass.

```
function m_b = bw(s,x,pins,pdel,pcopy)
N = length(s);
L = length(x)/2;
m_b = -realmax*ones(N,2*L);

logpins = log(pins);
logpdel = log(pdel);
logpcopy = log(pcopy);
for a=1:4
    for b=1:4
        logmut(a,b) = log(0.99)*(a==b) + log(0.01)*(a~=b);
    end
end

m_b(:,2*L) = 0;
for i=2*L-1:-1:1
    i
    for next=1:N
```

```
        if i~=L
            % insert
            vala = m_b(next,i);
            valb = m_b(next,i+1)+ logmut(s(next),x(i+1)) + logpins;
            m_b(next,i) = logsum([vala valb]);
            if next-2>=1
                % delete
                vala = m_b(next-2,i);
                valb  =  m_b(next-2,i+1) + logmut(s(next-2),x(i+1)) + logpdel;
                m_b(next-2,i) = logsum([vala valb]);
            end
            if next-1>=1
                % copy
                vala = m_b(next-1,i);
                valb  =  m_b(next-1,i+1) + logmut(s(next-1),x(i+1)) + logpcopy;
                m_b(next-1,i) = logsum([vala valb]);
            end
        else
            if next-90>=1
                for prev=max(1,next-110):next-90
                    vala = m_b(prev,i);
                    valb  =  logProbTruncPoiss(m_b(prev+1,i) - m_b(prev, i), 100);
                    m_b(prev,i) = logsum([vala valb]);
                end
            end
        end
    end
end

m_b = bw(seq,x,0.005,0.005,0.99);
logProb = logsum(m_b(:,end))
```

The resulting `logProb` is 8.2943.

To check your implementation run following code

```
m_f = fw(seq,x,0.005,0.005,0.99);
m_b = bw(seq,x,0.005,0.005,0.99);
mm = m_f + m_b;
logsum(mm(:,1))
logsum(mm(:,end))
```

If the two logsum calls output different values, you have a bug.

```
ans = -144.8799
ans = -141.1672
```

Problem 3(3pt) Implement Viterbi forward and backward pass. To do this transform your forward pass and backward by replacing the `logsum` with `max`

```
Viterbi Forward pass
=======================
function  m_f  =  vit_fw(s,x,pins,pdel,pcopy)
    N  =  length(s);
    L  =  length(x)/2;
    m_f  =  -realmax*ones(N,2*L);
    logpins  =  log(pins);
    logpdel  =  log(pdel);
    logpcopy  =  log(pcopy);

    for  a=1:4
        for  b=1:4
            logmut(a,b)  =  log(0.99)*(a==b)  +  log(0.01)*(a~=b);
        end
    end

    m_f(1:N,1)  =  -log(N)  +  logmut(s,x(1));
    for  i=2:2*L
        test = 1;
        for  prev=1:N
            if  i~=L+1
                %  insert
                vala  =  m_f(prev,i);
                valb  =  m_f(prev,i-1) +  logmut(s(prev),x(i))  +  logpins;
                %transform logsum to max
                m_f(prev,i) = max(vala, valb);

                if  prev<=N-2
                    %  delete
                    vala  =  m_f(prev+2,i);
                    valb  =  m_f(prev+2,i-1) + logmut(s(prev+2), x(i)) + logpdel;
                    %transform logsum to max
                    m_f(prev+2,i) = max(vala, valb);
                end

                if  prev<=N-1
                    %  copy
                    vala  =  m_f(prev+1,i);
                    valb  =  m_f(prev+1,i-1) + logmut(s(prev+1), x(i)) + logpcopy;
                    %transform logsum to max
                    m_f(prev+1,i)  =  max(vala, valb);
                end
            else
```

```
                if  prev+90<=N-1
                    for  next = prev+90 : min(prev+110,N-1)
                        vala  =  m_f(next,i);
                        diff  = m_f(next+1, i) - m_f(next, i);
                        valb  =  logProbTruncPoiss(m_f(next+1, i) - m_f(next, i), 100);

                        %transform logsum to max
                        m_f(next,i)  =  max(vala, valb);
                    end
                end
            end
        end
    end
end

function  logProb  =  logProbTruncPoiss(i,lambda)
    logProb  =  log(lambda)*i  +  (-lambda)  -  sum(log(1:i));
end

Viterbi Backward pass
=======================
function  m_b  =  vit_bw(s,x,pins,pdel,pcopy)
    N  =  length(s);
    L  =  length(x)/2;
    m_b  =  -realmax*ones(N,2*L);
    logpins  =  log(pins);
    logpdel  =  log(pdel);
    logpcopy  =  log(pcopy);
    for  a=1:4
        for  b=1:4
            logmut(a,b)  =  log(0.99)*(a==b)  +  log(0.01)*(a~=b);
        end
    end
    m_b(:,2*L)  =  0;
    for  i=2*L-1:-1:1
        for  next=1:N
            if  i~=L
                %  insert
                vala  =  m_b(next,i);
                valb  =  m_b(next,i+1)+  logmut(s(next),x(i+1))  +  logpins;
                %transform logsum to max
                m_b(next,i)  =  max(vala,  valb);

                if  next-2>=1
                    %  delete
                    vala  =  m_b(next-2,i);
```

```
                valb  =  m_b(next-2,i+1) + logmut(s(next-2),x(i+1)) + logpdel;
                %transform logsum to max
                m_b(next-2,i)  =  max(vala, valb);
            end
            if  next-1>=1
                %  copy
                vala  =  m_b(next-1,i);
                valb  =  m_b(next-1,i+1) + logmut(s(next-1),x(i+1)) + logpcopy;
                %transform logsum to max
                m_b(next-1,i)  =  max(vala, valb);
            end
        else
            if  next-90>=1
                for  prev=max(1,next-110):next-90
                    vala  =  m_b(prev,i);
                    diff = m_b(prev+1,i) - m_b(prev, i);
                    valb  =  logProbTruncPoiss(m_b(prev+1,i) - m_b(prev, i), 100);
                    %transform logsum to max
                    m_b(prev,i)  =  max(vala, valb);
                end
            end
        end
    end
  end
end

prob3.m
========
clear;
load('hw3.mat');
m_f  =  vit_fw(seq,x,0.005,0.005,0.99);
m_b  =  vit_bw(seq,x,0.005,0.005,0.99);
mm  =  m_f  +  m_b;
hMAP = zeros(length(x), 1);
for l = 1 : length(x)
    [maxVal,maxIndex] = max(m_b(l,:) + m_f(l,:));
    hMAP(l) = maxIndex;
end
```

Problem 4(2pt) Use MATLAB commands `tic` and `toc` to measure the time that it takes to run standard forward pass and Viterbi forward pass to complete. The ratio of the these times is 4.6450. Is one of these functions faster? If so, why?

Hint: One way to answer this question is to use MATLAB's profiler. Before calling a function you want to profile do folowing

```
profile clear
profile on
```

Run your code and once done (or once you interrupt it)

```
profile viewer
```

Rest ought to be self-explanatory.

Ans. Yes, the Viterbi forward pass is faster than the Standard forward pass. On profiling the standard forward pass, it is observed that most of the time is consumed by the *logsum* function defined as below:-

```
1 function  s  =  logsum(vec)
2    m  =  max(vec);
3    s  =  log(sum(exp(vec  -  m)))  +  m;
4 end
```

Line 3 of the above function "$s = log(sum(exp(vec - m))) + m$" is the killer time consumer as it involves computing the logarithmic sum of exponentiated values of the vector.

In contrast, the viterbi forward pass replaces the **logsum** computation with a simple **max(a, b)** function which is computationally less intensive and hence is a lot faster.

Problem 5(3pt) Modify your Viterbi forward pass to also record which of the states h_{i-1} was the most likely to have given rise to the current state h_i. This is sometimes called trace or backward pointers.

Implement code that starting with the state h_L with highest probability in the forward pass, backtracks according to the trace recording the path. Apply this procedure to long sequence `seq` and short sequence `x` in `hw3.mat`.

Paste most likely sequence of offsets in `seq` used to generate `x`

```
...
```