Numerically Stable Hidden Markov Model Implementation

Tobias P. Mann

February 21, 2006

Abstract

Application of Hidden Markov Models to long observation sequences entails the computation of extremely small probabilities. These probabilities introduce numerical instability in the computations used to determine the probability of an observed sequence given a model, the most likely sequence of states, and the maximum likelihood model updates given an observation sequence. This paper explains how to handle small probabilities by working with the logarithms of probabilities, rather than resorting to alternative rescaling procedures.

1 Introduction

A practical issue in the use of Hidden Markov Models (HMMs) to model long sequences is the numerical scaling of conditional probabilities. Conditional probabilities must be computed in order to efficiently estimate the most probable sequence of states for a model given some data. Conditional probabilities are also computed in the process of estimating HMM parameters given training data. The numerical issue arising from the computation of conditional probabilities is that the probability of observing a long sequence given most models is extremely small. The use of these extremely small numbers in computations leads to numerical instability, and makes application of HMMs to genome length sequences challenging. There are two common approaches to dealing with small conditional probabilities. One approach is to rescale the conditional probabilities using carefully designed scaling factors. The other approach is to work with the logarithms of the conditional probabilities. This paper explains how to implement the second approach, and argues that computing with logarithms has advantages over the scaling factor approach.

A common approach to eliminating numerical problems is to rescale the conditional probabilities by computing scaling factors. These scaling factors are designed to bring various conditional probabilities to within a range easily handled by standard machine floating point representation. In an excellent tutorial on HMMs, Rabiner [1] notes the potential numerical problems and outlines how to compute scaling factors, but the applicaton of these scaling factors is incompletely specified. A slightly different scaling factor derivation is provided in another nice tutorial by Minka [2], but Minka doesn't provide details on computing Baum Welch updates to estimate HMM parameters. Rescaling approaches offer a solution to the numerical instability of computing with very small conditional probabilities, but they also make the resulting code more complicated to understand and debug.

An alternative to the rescaling approach is to compute the logarithms of the conditional probabilities [1]. Working with logarithms has the advantage that scaling constants can be eliminated and Baum Welch parameter updates do not need to be re-derived; in addition,

[1] This is suggested on page 78 of Durbin et al [3]

numerical checks are easier to design using an intuitive understanding of the conditional probabilities than artificial scaled versions of the probabilities of interest. Rabiner provides a guide to a logarithmic implementation of the Viterbi algorithm in his tutorial, but does not explain how to implement the forward and backward algorithms, or the Baum-Welch parameter estimation algorithms using logarithms. Durbin et al allude to the technique necessary to work with logarithms, but the technique is again incompletely explained.

This paper proceeds by first defining functions necessary to compute hmm probabiliites using logarithms, and then provides pseudo-code to compute the main equations in Rabiner [1] using the logarithm technique suggested in Durbin et al. I conclude by arguing that the logarithm approach is both more convenient and easier to test than the scaling factor approach, although the logarithm approach is less efficient than the scaling approach.

2 Functions necessary for computation of probability logarithms

In order to implement the basic algorithms in Rabiner's tutorial using logarithms, we will need four functions and a value to represent the logarithm of zero. The four functions are essentially standard logarithm operations extended to correctly handle zero values; zero values must be handled because events can have zero probability. These functions will be used to compute the sums and products in Rabiner's tutorial while avoiding problems incured by small probabilities.

The logarithm of zero is not a number; the value representing the logarithm of zero in the implementation proposed here is best represented as the "Not a Number" value provided by most implementations of floating point arithmetic. I will write $LOGZERO$ to refer to this value.

The four functions we will use are an extended exponential function, extended logarithm function, a function for computing the logarithm of a sum given the logarithms of the summands, and a function for computing the logarithm of a product given the logarithms of the factors. I will use the prefix 'e' to distinguish the extended functions below from the standard functions.

- **eexp(x)** The extended exponential function is the standard exponential function $e^(x)$, except that it is extended to handle log zero, and is defined as follows.

  1. for a real number x:
  $$eexp(x) = e^{(x)} \tag{1}$$
  2. for $x = LOGZERO$
  $$eexp(x) = 0 \tag{2}$$

- **eln(x)** The extended logarithm is the standard logarithm provided by most floating point math libraries, except that it is extended to handle inputs of zero, and is defined as follows.

  1. for positive real numbers x:
  $$eln(x) = ln(x) \tag{3}$$
  2. for $x = 0$
  $$eln(x) = LOGZERO \tag{4}$$

- **elnsum(eln(x),eln(y))** The extended logarithm sum function computes the extended logarithm of the sum of x and y given as inputs the extended logarithm of x and y, and is defined as follows.

1. for positive real x and y:

$$elnsum(eln(x), eln(y)) = eln(x + y) \tag{5}$$

2. for $x = 0$

$$elnsum(LOGZERO, eln(y)) = eln(y) \tag{6}$$

3. for $y = 0$

$$elnsum(eln(x), LOGZERO) = eln(x) \tag{7}$$

- **elnproduct(eln(x),eln(y))** The extended logarithm product function returns the logarithm of the product of x and y.

1. for positive real x and y:

$$elnproduct(eln(x), eln(y)) = eln(x) + eln(y) \tag{8}$$

2. for $x = 0$

$$elnproduct(LOGZERO, eln(y)) = LOGZERO \tag{9}$$

3. for $y = 0$

$$elnproduct(eln(x), LOGZERO) = LOGZERO \tag{10}$$

Using these three functions, the various sums and products that must be computed for HMM use can be implemented simply, as shown by the pseudocode in the next section.

3 Pseudocode for log space computations

I begin with pseudo-code for the extended logarithm functions defined above, and then I move on to provide pseudo-code for the important operations in the Rabiner tutorial, where I use his nomenclature and provide pseudocode to solve Rabiner's problems one through three.

3.1 Extended logarithm functions

The floating point value "Not a Number" (or NaN) is a natural choice for implementing the value of $LOGZERO$. However, in many standard floating point implementations, any comparison involving NaN will return false. Most floating point implementations provide special functions for identifying not-a-number values (such as "isnan" in C) that must be used instead of the usual equality test.

The four functions below are extended exponential, extended logarithm, extended logarithm sum, and extended logarithm product.

Algorithm 1 Compute eexp(x)

```
if x = LOGZERO then
  0
else
  exp(x)
end if
```

Algorithm 2 Compute eln(x)

if $x = 0$ **then**
  $LOGZERO$
else if $x > 0$ **then**
  $ln(x)$
else
  negative input error
end if

Algorithm 3 Compute elnsum(eln(x),eln(y))

if eln(x)=LOGZERO **OR** eln(y)=LOGZERO **then**
  if eln(x)=LOGZERO **then**
    eln(y)
  else
    eln(x)
  end if
else
  if eln(x) > eln(y) **then**
    eln(x) + eln(1+exp(eln(y)-eln(x)))
  else
    eln(y) + eln(1+exp(eln(x)-eln(y)))
  end if
end if

The formula for computing elnsum can be derived as follows.

$$\begin{aligned} ln(x) + (ln(1 + e^{ln(y)-ln(x)})) &= ln(x) + ln\big(1 + e^{ln(y/x)}\big) \\ &= ln(x) + ln\Big(1 + \frac{y}{x}\Big) \\ &= ln(x) + ln\Big(\frac{x+y}{x}\Big) \\ &= ln(x) + ln(x+y) - ln(x) \\ &= ln(x+y) \end{aligned} \tag{11}$$

It is important to check the size of the logarithms of the summands, because the numerical stability of this computation depends on small values of the exponential term in $ln(1 + e^{ln(y)-ln(x)})$. Small values can be ensured by always subtracting the larger argument from the smaller argument so that the sign of the difference of logarithms is negative. For example, consider the case where the summands are e^{60} and e^{-10}. The logarithms of the summands are 60 and -10, and the logarithm of the sum can be computed either as $ln(e^{60}) + ln(1 + e^{-70})$, or as $ln(e^{-10}) + ln(1 + e^{70})$. In the first alternative, the quantity e^{-70} gets underflowed to zero, and the result is 60, which is as accurate as most machine precisions will allow. In the second alternative, e^{70} will evaluate to inf, the floating point constant used to represent a number to large to be represented in the machine precision, and the numerical value of the computation will be lost. When computing the logarithm of the sum of two numbers that have substantially different magnitudes, the exact form of the computation employed can make the difference between a usable result and an overflow error.

Algorithm 4 Compute elnproduct(eln(x),eln(y))

if eln(x)=LOGZERO **OR** eln(y)=LOGZERO **then**
  LOGZERO
else
  eln(x)+eln(y)
end if

3.2 Problem 1: Computing the Probability of Observing a Sequence

I am assuming familiarity with the Rabiner tutorial and will use his notation and definitions.

In the forward algorithm, the object is to compute the joint probability that the partial sequence from time 1 to t was observed and that the state of the Markov process was S_j at time t, given a model λ; this is writte $\alpha_t(j)$ and is defined (Rabiner's equations 18-20) as

$$\begin{aligned}\alpha_t(j) &= P(O_1 O_2 \cdots O_t, q_t = S_j|\lambda) \\ &= \pi_j b_j(O_1) \ for\ t = 1 \\ &= b_j(O_t) \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \ for\ t > 1\end{aligned} \tag{12}$$

In the log space forward algorithm, the object is simply to compute $ln(\alpha_t(i))$, which will be written $eln\alpha_t(i)$.

Algorithm 5 Compute $eln\alpha_t(i)$ for all states S_i and observations O_t

for $i = 1$ to N **do**
  $eln\alpha_1(i) \leftarrow elnproduct(eln(\pi_i), eln(b_i(O_1)))$
end for
for $t = 2$ to T **do**
  for $j = 1$ to N **do**
    $logalpha \leftarrow LOGZERO$
    for $i = 1$ to N **do**
      $logalpha \leftarrow elnsum(logalpha, elnproduct(eln\alpha_{t-1}(i), eln(a_{ij})))$
    end for
    $eln\alpha_t(j) \leftarrow elnproduct(logalpha, eln(b_j(O_t)))$
  end for
end for

In the backward algorithm, the object is to compute the joint probability of observing the partial sequence from observation $t + 1$ to T given the model λ and that the hidden state is S_i at time t; this is written $\beta_t(i)$ and is defined (Rabiner's equations 23-25) as

$$\begin{aligned}\beta_t(i) &= P(O_{t+1} O_{t+2} \cdots O_T|q_t = S_i, \lambda) \\ &= 1 \ for\ t = T \\ &= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \ for\ 1 \le t < T\end{aligned} \tag{13}$$

In the log space backward algorithm, the object is to compute $ln(\beta_t(i))$, which will be written $eln\beta_t(i)$.

Algorithm 6 Compute $eln\beta_t(i)$ for all states S_i and observations O_t

for $i = 1$ to N **do**
  $eln\beta_T(i) \leftarrow 0$
end for
for $t = T - 1$ to 1 **do**
  for $i = 1$ to N **do**
    $logbeta \leftarrow LOGZERO$
    for $j = 1$ to N **do**
      $logbeta \leftarrow elnsum(logbeta, elnproduct(eln(a_{ij}),$
        $elnproduct(b_j(O_{t+1}), eln\beta_{t+1}(j))))$
    end for
    $eln\beta_t(i) \leftarrow logbeta$
  end for
end for

3.3 Problem 2: Computation of state probabilities given a model and a sequence

Rabiner already provides the log space method for the viterbi algorithm (his equations 104-105c). Therefore, I will only provide pseudocode for computing $\gamma_t(i)$, which is the probability of being in state S_i at time t given the model and observation sequence, and is defined (Rabiner's equation 27, but with the denominator summation variable changed) as:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{14}$$

This section shows how to compute the log of gamma, written $eln\gamma_t(i)$, presuming the log versions of the forward and backward computations are completed.

Algorithm 7 Compute $eln\gamma_t(i)$ for all states S_i and observations O_t

Require: The log space forward and backward variables have been computed
for $t = 1$ to T **do**
  $normalizer \leftarrow LOGZERO$
  for $i = 1$ to N **do**
    $eln\gamma_t(i) \leftarrow elnproduct(eln\alpha_t(i), eln\beta_t(i))$
    $normalizer \leftarrow elnsum(normalizer, eln\gamma_t(i))$
  end for
  for $i = 1$ to N **do**
    $eln\gamma_t(i) \leftarrow elnproduct(eln\gamma_t(i), -normalizer)$
  end for
end for

3.4 Problem 3: Baum-Welch updates of model parameters

In order to compute revised model parameters, we need to compute the state probabilities of problem 2, as well as a new probability $\xi_t(i, j)$, which is defined as the probability of being in state S_i at time t and state S_j at time $t + 1$, given a model λ and observation sequence O. Rabiner's equations 36 and 37 (where the variables of summation in the denominator have again been changed) are:

$$\begin{aligned}\xi_t(i,j) &= P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \\ &= \frac{\alpha_t(j) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{k=1}^{N} \sum_{l=1}^{N} \alpha_t(k) a_{kl} b_l(O_{t+1}) \beta_{t+1}(l)}\end{aligned} \tag{15}$$

I will write $eln\xi_t(i,j)$ for the log space version of this probability.

Algorithm 8 Compute $eln\xi_t(i,j)$ for all state pairs S_i and S_j and observations O_t

Require: The log space forward and backward variables have been computed
  for $t = 1$ to $T - 1$ **do**
    $normalizer \leftarrow LOGZERO$
    for $i = 1$ to N **do**
      for $j = 1$ to N **do**
        $eln\xi_t(i,j) \leftarrow elnproduct(eln\alpha_t(i), elnproduct(eln(a_{ij}),$
            $elnproduct(eln(b_j(O_{t+1})), eln\beta_{t+1}(j))))$
        $normalizer \leftarrow elnsum(normalizer, eln\xi_t(i,j))$
      end for
    end for
    for $i = 1$ to N **do**
      for $j = 1$ to N **do**
        $eln\xi_t(i,j) \leftarrow elnproduct(eln\xi_t(i,j), -normalizer)$
      end for
    end for
  end for

Once the $eln\gamma$ and $eln\xi$ variables are computed, the updated parameters can be computed.

The updated state probabilities are simply the extended exponentials of the values of $eln\gamma_1(i)$ (Rabiner's equation 40a).

Algorithm 9 Compute π_i, the estimated probability of starting in state S_i

Require: The log space variable $eln\gamma$ has been computed
  $\pi_i \leftarrow eexp(eln\gamma_1(i))$

The updated transition probabilities are computed using both the $eln\gamma_t(i)$ and $eln\xi_t(i,j)$ variables.

Algorithm 10 Compute a_{ij}, the estimated probability of transitioning from state S_i to S_j

Require: The $eln\gamma$ and $eln\xi$ variables have been computed
  $numerator \leftarrow LOGZERO$
  $denominator \leftarrow LOGZERO$
  for $t = 1$ to $T - 1$ **do**
    $numerator \leftarrow elnsum(numerator, eln\xi_t(i,j))$
    $denominator \leftarrow elnsum(denominator, eln\gamma_t(i))$
  end for
  $a_{ij} \leftarrow eexp(elnproduct(numerator, -denominator))$

Finally, the updated emission probabilities are computed using the $eln\gamma_t(i)$ variables, the $eln\xi_t(i,j)$ variables, and finally the observation sequence O_t.

Algorithm 11 Compute $b_j(k)$, the estimated probability of emitting symbol v_k from state S_j

Require: The $eln\gamma$ and $eln\xi$ variables have been computed
  $numerator \leftarrow LOGZERO$
  $denominator \leftarrow LOGZERO$
  for $t = 1$ to T **do**
    if $O_t = v_k$ **then**
      $numerator \leftarrow elnsum(numerator, eln\gamma_t(j))$
    end if
    $denominator \leftarrow elnsum(denominator, eln\gamma_t(j))$
  end for
  $b_j(k) \leftarrow eexp(elnproduct(numerator, -denominator))$

4 Conclusions

This paper has presented an alternative to Rabiner's scaling approach for handling numerical instability in HMM computations. The probability logarithm approach is simple, and has the advantage that probabilities of probability measures can be used to check the code at intermediate stages. For instance, the sum over all states S_i of $\gamma_t(i)$ should be one, and thus the logarithm of the sum of all states of $\gamma_t(i)$ should be zero. Similarly, the sum over all state pairs S_i and S_j of $\xi_t(i, j)$ should be one. These properties can be checked with the log sum function.

The approach presented here is similar in spirit to Rabiner's method for using logarithms to modify the Viterbi algorithm for numerical stability. The logarithm approach has two advantages. First, no derivations to use the scaled variables need be done - Rabiner's formulas can be used almost directly. Second, it is easy to design checks into the code to verify the accuracy of the computations.

Rabiner's scaling method is more efficient, because the logarithm approach requires computing a logarithm and an exponential for every addition. The logarithm and the exponential function are both expensive to compute. The use of tables could speed up the logarithm and exponential computations, although at a loss of accuracy. For application of an HMM with a small number of states to a relatively small chromosome, however, the logarithm approach is fast enough in practice.

5 References

[1] Rabiner L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proc. IEEE, Vol. 77, No. 2, 1989
[2] Minka T.P. From Hidden Markov Models to Linear Dynamical Systems.
http://citeseer.ist.psu.edu/minka99from.html, 1999.
[3] Durbin R., Eddy S., Krogh A., Mitchison G. Biological Sequence Analysis. Cambridge University Press, 1998.