3.6 Numerical stability of HMM algorithms

Even on modern floating point processors we will run into numerical problems when multiplying many probabilities in the Viterbi, forward, or backward algorithms. For DNA for instance, we might want to model genomic sequences of 100 000 bases or more. Assuming that the product of one emission and one transition probability is typically 0.1, the probability of the Viterbi path would then be of the order of $10^{-100\,000}$. Most computers would behave badly with such numbers: either an underflow error would occur and the program would crash; or, worse, the program would keep running and produce arbitrary wrong numbers. There are two different ways of dealing with this problem.

The log transformation

For the Viterbi algorithm we should always use the logarithm of all probabilities. Since the log of a product is the sum of the logs, all the products are turned into sums. Assuming the logarithm base 10, the log of the above probability of $10^{-100\,000}$ is just $-100\,000$. Thus, the underflow problem is essentially solved. Additionally, the sum operation is faster on some computers than the product, so on these computers the algorithm will also run faster.

We will put a tilde on all the model parameters after taking the log, so for example $\tilde{a}_{kl} = \log a_{kl}$. Then the recursion relation for the Viterbi algorithm (3.8) becomes

$$V_l(i+1) = \tilde{e}_l(x_{i+1}) + \max_k (V_k(i) + \tilde{a}_{kl}),$$

where we use V for the logarithm of v. The base of the logarithm is not important as long as it is larger than 1 (such as 2, e, and 10).

It is more efficient to take the log of all the model parameters before running

the Viterbi algorithm, to avoid calling the logarithm function repeatedly during the dynamic programming iteration.

For the forward and backward algorithms there is a problem with the log transformation: the logarithm of a sum of probabilities cannot be calculated from the logs of the probabilities without using exponentiation and log functions, which are computationally expensive. However, the situation is not in practice so bad. Assume you want to calculate $\tilde{r} = \log(p+q)$ from the log of the probabilities, $\tilde{p} = \log p$ and $\tilde{q} = \log q$. The direct way is to do $\tilde{r} = \log(\exp(\tilde{p}) + \exp(\tilde{q}))$. By pulling out $\tilde{p}$, one can write this as

$$\tilde{r} = \tilde{p} + \log(1 + \exp(\tilde{q} - \tilde{p})).$$

It is possible to approximate the function $\log(1+\exp(x))$ by interpolation from a table. For a reasonable level of accuracy, the table can actually be quite small, assuming we always pull out the largest of $\tilde{p}$ and $\tilde{q}$, because $\exp(\tilde{q} - \tilde{p})$ rapidly approaches zero for large $(\tilde{p} - \tilde{q})$.

Scaling of probabilities

An alternative to using the log transformation is to rescale the f and b variables, so they stay within a manageable numerical interval [Rabiner 1989]. For each i define a scaling variable s_i, and define new f variables

$$\tilde{f}_l(i) = \frac{f_l(i)}{\prod_{j=1}^{i} s_j}. \tag{3.26}$$

From this it is easy to see that

$$\tilde{f}_l(i+1) = \frac{1}{s_{i+1}} e_l(x_{i+1}) \sum_k \tilde{f}_k(i) a_{kl},$$

so the forward recursion (3.11) is only changed slightly. This will work however we define s_i, but a convenient choice is one that makes $\sum_l \tilde{f}_l(i) = 1$, which means that

$$s_{i+1} = \sum_l e_l(x_{i+1}) \sum_k \tilde{f}_k(i) a_{kl}.$$

The b variables have to be scaled with the same numbers, so the recursion step in (3.3) becomes

$$\tilde{b}_k(i) = \frac{1}{s_i} \sum_l a_{kl} \tilde{b}_l(i+1) e_l(x_{i+1})$$

This scaling method normally works well, but in models with many silent states, such as the one we describe in Chapter 5, underflow errors can still occur.