# *Dynamic Programming for HMMs*

Bernard M.E. Moret

Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of states of the HMM and let $\Sigma$ be its output alphabet. Let the state transition matrix be the $n \times n$ stochastic matrix $A = (a_{ij})$, where $a_{ij}$ is the probability of a transition from state $i$ to state $j$, and let the emission matrix be the $n \times |\Sigma|$ matrix $E = (e_{ij})$, where $e_{ij}$ is the probability of producing character $j$ when in state $s_i$. Denote by $\alpha \cdot n$ the number of nonzero entries in matrix $A$. Finally, denote by $x$ the observed output string, by $x_i$ the character in the $i$th position of $x$, and by $m$ the length of $x$.

We propose three related questions:

1. How do we reconstruct (and compute the likelihood of) the most likely state sequence for the given HMM and given output sequence?

2. How do we compute the likelihood of a given sequence?

3. How do we compute the likelihood of a particular state at a particular step for the observed output sequence?

All three of these questions can be answered by the same basic dynamic program, with small variations. We will build a table with one row for each state and one column for each position in the observed output sequence—thus our table will be $n \times m$. (Note that we may want to introduce two additional states, i.e., two additional rows, and one additional column; the extra states are a start state and an end state, the extra column corresponds to position 0—before the position of the first characters in the output string. These additions do not alter the following, but prevent having to treat special rows or columns differently from others and thus are likely to improve efficiency.)

1. The solution here is the so-called Viterbi algorithm. We will fill the table column by column, starting at column 0 and ending at column $m$; entry $ij$ in the table, that is, $T(i, j)$, will denote the probability of the most likely path to state $s_i$ after reading the $j$th character in $x$. We can compute $T(i, j)$ according to the following recurrence:

   $$T_v(i, j) = \max_l \left( a_{li} \cdot e_{i,x_j} \cdot T_v(l, j-1) \right)$$

   In words, we just look one step back along the path, to a previous entry $T_v(l, j-1)$ for some previous state $l$, and compute the probability of the extension from that previous step to the current state $s_i$ at step $j$ with symbol $x_j$ emitted, retaining the largest one.

   It should be noted, that, in spite of appearances, what is computed is really a joint probability, not (at least not intentionally) a conditional probability. That is, the recurrence computes the maximum value of the probability $P(\pi, x)$, where $x$ is the output string and $\pi$ is a path through the states of the HMM that produces that string. However, it is not hard to see that computing the max (over choices of $\pi$) of $P(\pi, x)$ is equivalent to computing the max of $P(\pi \mid x)$, the conditional (posterior) probability.

2. To get the likelihood of the output sequence, we could look at every state path that can generate it, compute the probability of each such path, then sum them all to obtain $P(x) = \sum_\pi P(\pi, x)$, using the notation from (i); but this takes time proportional to the number of paths and thus could take exponential time. Instead, we implicitly look at all paths every

time we add another output character. The recurrence is nearly identical to the Viterbi recurrence—we simply replace the max operator by an addition.

$$T_f(i,j) = \sum_l \big(a_{li} \cdot e_{i,x_j} \cdot T_f(l, j-1)\big)$$

(Note that the emission probability is independent of the summation index and so can be factored.) The dynamic program resulting from this recurrence is often called the forward algorithm, hence my choice of $T_f$ for this function.

3. Here we really want to compute $P(s_i, j \mid x)$, the probability that, given that the HMM produced the output sequence $x$, it was in state $s_i$ at step $j$. Because this is a conditional probability (unlike the two computed above), we proceed somewhat indirectly, using joint probabilities that we do know how to compute. We have

$$P(s_i, j, x) = P(s_i, x_1 x_2 \cdots x_j) \cdot P(x_{j+1} x_{j+2} \cdots x_m \mid s_i, x_1 x_2 \cdots x_j)$$

Note that the first term in the product is exactly $T(i,j)$ in the forward algorithm, so we know how to compute it. The second term can be simplified: thanks to the Markov (memoryless) property, the dependency on $x_1 x_2 \cdots x_j$ does not exist, so the second term now reduces to $P(x_{j+1} x_{j+2} \cdots x_m \mid s_i)$, and we can easily compute that term by the same dynamic program again, but this time running it backward (which gives us the condition for free). The recurrence is then

$$T_b(i,j) = \sum_l a_{li} \cdot e_{i,x_{j+1}} \cdot T_b(l, j+1)$$

where the $b$ subscript denotes that the backward version. Now we simply have

$$P(s_i, j \mid x) = \frac{T_f(i,j) \cdot T_b(i,j)}{P(x)}$$

What is the running time of each of these three algorithms? All have to fill in the entire $n \times m$ table; moreover, in order to fill it, all have to look at all transitions into (or out of) each state. Thus, to process one column of the table, all transitions of the HMM must be examined and used in the recurrence, so that the cost of processing one entire column is proportional to $\alpha \cdot n$, the number of nonzero entries in the transition matrix. Thus the overall running time of the algorithms is $\Theta(\alpha \cdot n \cdot m)$; this can be as low as order $mn$ if $\alpha$ (which is twice the average degree of a node in the state transition diagram) is a constant and as high as order $mn^2$ if $\alpha$ is some fraction of $n$ (as in a dense state graph).