To illustrate the Forward-Backward algorithm, let’s start from the analytic equations.

Denote the observations as , parameter , where denotes the matrix (n\*m) containing all the conditional probabilities of observations given each state at each timestamp (argument obs in the function). denotes the sequence of length m storing all the floating-point timestamps (argument ts in the function).

Similarly with the lecture notes, , except the representation of timestamps is modified due to the continuous situation.

The transitional probabilities are , so there are only 2 situations for the probability given the timestamp for transition, and they only rely on the parameter and the values of the adjacent timestamps. Denote the first probability as , the second probability as . This simple format of transitional probabilities gives us chances to reduce the time complexity. To clarify:

Firstly, (1), and

For this forward part, since I implemented the algorithm using iterations, all the values of the previous layers are stored in a matrix alpha, so we can use all the values at layer to calculate the first value of layer based on the equation (1) above (time complexity O(n)). Then we can calculate the other values of layer based on the first value with the equation (2), with , already calculated and stored (time complexity O(n) in total). Therefore, we get all the k-th layer with time complexity O(2n), so we can calculate all the alpha matrix with O(mn).

Secondly, (1),

With the same manner, the backward part’s matrix beta can also be calculated in O(mn).

All in all, the forward-backward algorithm is realized with time complexity O(mn) in this way.

For the Viterbi path algorithm in time complexity of O(nm).

With the same notation defined above, (1)

Since and are used commonly by all the values at level , we can use one iteration of size n to store all the values of for each into a vector, and the maximum and 2nd maximum (denote as , ) values and Ids (denote as ) of , . Then we use another iteration of size n to populate all the values of . For each iteration, if it’s , we calculate the maximum of and , and multiply it with as the value of . Otherwise, we calculate the maximum of and , and multiply it with as the value of , . Within each iteration, the time complexity is constant. Therefore, we used O(n) to populate the k-th level from the (k-1)-th level, and store the Ids into phi.

Based on this thinking, I rewrite the original Viterbi algorithm as calculating all the values of a level if its empty. Although there’s still many redundant function calls left for each element due to the function interface design, it doesn’t run into detailed calculation so being constant consumption, so we still have O(n) for each level, and O(nm) in total. There are some ways to upgrade by exporting vector for each function call, so that unnecessary function calls can be relieved, or just using iterations to implement.

For maintaining their functionality when scaling to large dimensions, I used two methods. I used logarithm for the first one of Viterbi path, because the elements of matrix delta are not necessary to be converted back to determine the path, and additions are more efficient than multiplication. Therefore, using log-transformation for the Viterbi algorithm is a good choice. I used another method for the forward-backward for its special properties.

After noticing for all these three algorithms they have a trend of decreasing towards the direction of calculation with some speed, I think we can scale up a little in each level of calculation to keep the elements staying within the range of double type (). Of course this is also based on the property that the results we needed for the forward-backward and Viterbi algorithms are all not related to the scaling in each level. I used 1.1 times (1.1^5000=1e206) of scaling up in the forward-backward algorithm, and in the result of testing all the elements of alpha and beta matrices can stay within the range of 1e-100 and 1e100, which are still far from the boundary. The addition and division in the main function would automatically remove the scaling effect and give us the correct result. This method can also be used to the Viterbi path algorithm.