CSC412 – Assignment #4

Due: March29, 2004 in class
Worth: 15%
Late assignments not accepted.

1 Training a Profile HMM to align DNA sequences

In this assignment, you will train a Profile HMM to align several fragments of DNA sequences representing various versions of the haemogloben protein coding region. You will train the model using maximum likelihood as a cost function and the EM algorithm as an optimizer.
The data consist of 59 subsequences, each of length 100 base pairs, over the alphabet $\{a, t, c, g\}$.

- As a reminder, the HMM is an unsupervised learning model for sequence data $Y = \{y_1 \dots y_T\}$ which uses a discrete hidden state sequence $X = \{x_1 \dots x_T\}$ to explain the observations.
  As you know, the joint probability over hidden states and observations factorizes as follows:

$$\begin{aligned} p(x_1 = i) &= \pi_i \\ p(x_t = j | x_{t-1} = i) &= S_{ij} \qquad t = 2 \dots T \\ p(y_t = m | x_t = k) &= A_{mk} \qquad t = 1 \dots T \end{aligned}$$

  where S is the state transition matrix, π governs the initial state distribution and the matrix A controls the output parameters of each state.

- The marginal probability of an observed sequence is given by summing over all possible state trajectories:

$$\log p(\{y_1, \dots y_T\}) = \log \sum_{\{x_1, \dots x_T\}} p(\{x_1, \dots x_T\}, \{y_1, \dots y_T\})$$

- The model is trained to maximize the log likelihood of a set of training sequences
  $\{y_1^1, \dots y_{T_1}^1\}, \dots, \{y_1^c, \dots y_{T_2}^c\}, \dots, \{y_1^C, \dots y_{T_N}^C\}$,
  given by the sum of the log likelihoods for each sequence.

2 Profile (String-Edit) HMMs

In a profile HMM, we learn an unkown "template" sequence which is being matched to each observation sequence using insertions, deletions, substitutions and matches. The hidden states come in three types: insert, delete and match/substitute. Each template position has associated with it exactly one insert state, one delete state, and one match state (which does matches and substitutions). We will also add a special "end state" after the last template position.

- There are exactly $3N + 1$ states (including the end state) in a profile HMM where N is the length of the template being assumed.

- Most of the transition probabilities are zero, since transitions are permitted only between states at the same position or neighbouring positions in the template.

- In this assignment, we will use a simplified version of the profile HMM in which all the transition probabilities depend only on the type of states between which the transition is occuring (but not on the position within the template). In particular, we will work with the following values:

$$\begin{aligned}
\pi_{match1} &= \alpha \\
\pi_{del1} = \pi_{ins1} &= (1-\alpha)/2 \\
S_{w_{n-1} \to match_n} &= \alpha \\
S_{w_{n-1} \to del_n} &= (1-\alpha)/2 \\
S_{w_n \to ins_n} &= (1-\alpha)/2 \\
S_{w_N \to END} &= \alpha + (1-\alpha)/2 \\
S_{END \to END} &= 1
\end{aligned}$$

  where w represents any of $\{match, ins, del\}$. Thus, the probability of transitioning *into* a match state is α no matter what state you are coming from, and the probability of transitioning *into* an insert or delete state is $(1-\alpha)/2$ regardless of the previous state.

- The END state can only be reached from the states in the last template position, N. Once the model enters the END state, it stays there forever. The output model for the end state outputs only the special end symbol, which you should append to the final position of each training sequence, to force the model to reach the end state after outputting the rest of the sequence.

3 What to do

Using the forward-backward and Baum-Welch equations for inference and learning, train a profile HMM on the DNA sequences provided. Of couse, you must make sure your equations correctly deal with the deletion states in the profile HMM and properly handle multiple training sequences. In particular:

- Use the 59 sequences of length 100 provided in the file `a4dna.mat` as your data.
- Use 100 template positions in the model, giving 300 states in all.
- Do not train the transition matrix S or the initial state distribution π or the output model for the END state. Put zeros in all the places where they belong, and use a value of $\alpha = 0.95$ for the nonzero transitions, as described above. During training, do not perform any updates on S or π or A_{END}. Since you are not updating the transitions, you won't need to compute any joint expectations of the form $\xi_{ij}(t) = p(x_t = i, x_{t+1} = j|Y)$. Instead you only need to compute the marginal expectations $\gamma_i(t) = p(x_t = i|Y)$.
- At each iteration, run the α and β recursions to compute the state expectations and the likelihood in the E-step, and update the output paramters A for the insert and match states using the Baum-Welch formulas for the M-step.
  (Remember, there are no output parameters for the delete states, and you are not updating S or π or A_{END}.)
- Continue iterating until convergence. You can assess convergence of EM by monitoring the log likelihood and stopping when the fractional change `(newlik-oldlik)/abs(newlik)` is very small (say less than one part in a hundred thousand.) Remember, your log likelihood should never decrease!
- Restart the training at several random initializations of the parameters, and run EM to convergence each time. Keep the parameter set which gave the best average training likelihood.

After training the model, do Viterbi (MAP) decoding on each training sequence to find the most likely state path through the model given the observation sequence. (Using the parameters from your best run.) This will tell you, for each symbol, whether the model used a deletion, insertion or a match to a template position (and if so which one) to generate it.

4 What to hand in

- A plot of the trajectory of the average training log likelihood per sequence for the run that resulted in the best parameters. Show iterations of EM horizontally and average log likelihood vertically.
  Clearly indicate the stopping criteria used to assess convergence of EM
- For each match state, find the most likely symbol to be output from that state. Display this sequence of 100 most likely symbols.
- A display of the aligned data, based on the MAP state estimate of each sequence, made as follows:
  When the model goes through a delete state, output a ':'. When the model goes through an insert state, output the corresponding symbol from the sequence as a capital letter, A/T/C/G. When the model goes through a match state, output the corresponding symbol from the sequence as a lowercase letter, a/t/c/g.
- Another display of the aligned data, based on the MAP state estimate of each sequence, made as follows:
  If the model went through the match state corresponding to template position n, print the corresponding symbol from the sequence in column n of the output. If the model went through the delete state and/or the insert state corresponding to template position n (but not the match state), print a space in column n of the output.

5 Some hints

- Initialze the output parameters of the insert and match states to be roughly uniform, plus enough noise to explore different local optima. Don't forget to renormalize the columns of A after you add noise.
- Be careful about numerical scaling. You can either use a rescaling trick to renormalize α and β as you go, or you can represent the log of α and β and use `logsum` when you need to update them in the recursions.
- Initializing the α recursion is a bit tricky. Think it over carefully, asking yourself, what states could the model end up in after generating only a single output symbol.
- When dealing with the END state,you can introduce a special "end" symbol, which appears exactly once in each training sequence, at the very last position (101). Then you can make a a special case in the piece of code that computes $A_{y(t),k}$ so that if $y(t)$ is the end symbol, then $A_{y(t),k}$ is either zero (if k is not the END state) or one (if k is the END state). $A_{y(t),k}$ should be zero if k is the END state and $y(t)$ is *not* the end symbol. Alternately, you can treat the end symbol to as a regular symbol, which would make the alphabet size 5 instead of 4. If you do this, make sure to initialize all states other than the end state to have *zero* probability of outputting the end symbol, and make sure to initialize the END state to have probability one of outputting the end symbol and probability zero of outputting any other symbol.

6 Optional Stuff

Something fun you can try, which you won't be graded on:

- Try redoing the first display of the algined data above so that whenever a symbol y_t^c in sequence c is generated from a particular match state, it appears on row c but in the same column of the ascii output as all other symbols generated from that same match state. This involves padding each output line with some spaces (or another blank character, such as '-') to get the symbols from various match states lined up properly. This effectively combines the two displays I asked you to make above into a single display which simultaneously shows exactly what alignment happend to each sequence and what the template looks like.