

Problem Set 2: Hidden Markov Models

Due at the beginning of class on Wednesday, February 11

Collaboration is allowed in groups of at most three students, but you must submit separate writeups. Please also write the names of all your collaborators on your submissions. If you are working alone, we will drop the problem with the lowest score.

Problem 1. (25 points) *HMMs and training*

(a) (6 points) Suppose we have a large number of sequences emitted by an HMM that has a particular transition probability $a_{kl} = 0$, for some k and l . Say that we now use these emitted sequences to train (using Baum-Welch) a new HMM with the same architecture, one that happens to start with $a_{kl} = 0$. Prove that the parameter a_{kl} will remain 0 after the training.

(b) (7 points) Assume we have an HMM to train, and we choose initial conditions such that two of the states, k and k' , are identical. That is, for all other states l , $a_{kl} = a_{k'l}$, $a_{lk} = a_{lk'}$, and for all characters b , $e_k(b) = e_{k'}(b)$. Prove or disprove that Baum-Welch will keep these two states identical. How about Viterbi training?

(c) (11 points) Initial parameters play an important role on the eventual result of HMM training. Both Baum-Welch and Viterbi training are, unfortunately, local optimizations. Prove this, using the following construction:

- i. Choose one algorithm: Baum-Welch or Viterbi.
- ii. Create an HMM which is the “true” model of a random process, such as a coin toss with fair and loaded dice.
- iii. Create a set of (one or more) training sequences. Their number and length is your choice, but they should satisfy one requirement: A_{kl} and $E_k(b)$, the counts of each transition and emission in all the sequences, should be exactly consistent with a_{kl} and $e_k(b)$, the true model parameters.

$$\forall k, l, b: a_{kl} = \frac{A_{kl}}{\sum_j A_{kj}} \text{ and } e_k(b) = \frac{E_k(b)}{\sum_x E_k(x)}$$

- iv. Demonstrate a set of initial nonzero parameters for a new HMM which has the same architecture, such that your chosen learning algorithm converges to the true model.
- v. Demonstrate another set of initial nonzero parameters, such that your chosen learning algorithm converges to a wrong local maximum.

Problem 2. (25 points) *HMMs and parsing*

(a) (5 points) Given an observed sequence x , we have seen an efficient method in class for calculating $\operatorname{argmax}_{\pi} P(\pi|x)$. One might ask why we are interested in this π and not $\operatorname{argmax}_{\pi} P(x|\pi)$ instead. The latter state sequence, after all, is the one that has the highest probability of producing x . Illustrate why this is the wrong π to find, with the help of an example HMM.

(b) (8 points) Our friend, the casino player, is again trying to find a way to take your money. This time, however, you are armed with your knowledge of HMMs and will not play a game unless the expected payoff to you is nonnegative. He is throwing a fair coin (50% H, 50% T) and a loaded coin (90% H, 10% T). He suggests the following game: he will throw 1000 rolls, starting with fair (50%) or loaded (50%) and exchanging between the two with probability 1% before each roll.

After you observe all 1000 rolls, which have been created by some true, unknown to you, sequence of states π^* , you are asked to make a guess π on which ones were fair (F), and which ones were loaded (L). For each roll you choose F, if you are correct you gain $\$C_F$, otherwise you lose $\$W_F$. Similarly, for each roll you choose L, you gain/lose $\$C_L$ and $\$W_L$, respectively. The following table summarizes the payoff:

π_i^*	π_i	$\text{Win}(\pi_i^*, \pi_i)$
F	F	$+C_F$
F	L	$-W_L$
L	F	$-W_F$
L	L	$+C_L$

Fortunately, you have your laptop with you and you can code up quickly any algorithm you like, run it, and devise the optimal strategy, i.e., which rolls to call fair and which loaded. Note that the optimal strategy is not necessarily to run Viterbi, which will give you the overall most likely parse. Instead, try to find the strategy that will maximize your expected payoff:

$$E_{\pi^*}[\sum_i \text{Win}(\pi_i^*, \pi_i)]$$

(c) This time the casino player is in trouble. After he rolls 1000 rolls, a fortune teller tells you the exact number of rolls that are loaded, namely 600. Can you take advantage of this information?

(i) (7 points) To begin with, devise an algorithm to compute for $1 \leq i \leq 1000$, $P(\pi_i^* = F \mid x, (\sum_j 1(\pi_j^* = L)) = 600)$. That is, we want the conditional posterior probabilities that any given state is F or L, given that we know the exact total number of fair and loaded states. What is the running time of your algorithm?

(ii) (5 points) What is the optimal parse for 1000 rolls now, i.e. the strategy that maximizes the expected payoff as in part (b). Are you going to guess 600 loaded rolls, or not necessarily?

Problem 3. (25 points) *Pair HMMs for Alignments—Architecture*

For the following problems, you may ignore start and end states, unless they are important for the function of the HMM. You do not need to give the exact probabilities for all edges and emissions, but you should explain how one could compute them from the parameters given.

(a) (6 points) What is the smallest pair HMM you can build that corresponds to running Needleman-Wunsch with constant (not affine) gap penalty (with parameters: match m , mismatch s , gap d).

(b) (6 points) Construct a pair HMM for *overlap-detection* alignment, as described in lecture 3. Show the state diagram with transition probabilities and explain briefly. Do not show any Viterbi recurrences.

(c) (6 points) Construct a pair HMM that performs regular Needleman-Wunsch alignment with affine gap penalty, but where the gaps can be of at most length L .

(d) (7 points) Construct a pair HMM that will perform alignment that is equivalent to a variant of Needleman-Wunsch with piecewise linear gaps, i.e., a gap of length k incurring penalty not $d+ek$, but $\min\{d_1 + e_1k, \dots, d_s + e_s k\}$, and which has at most $2s+1$ states.

Problem 4. (25 points) *Pair HMMs for Alignments—Parsing*

(a) (8 points) In aligning a pair of sequences $x = x_1x_2\dots x_n$ (of length n) and $y = y_1y_2\dots y_m$ (of length m), you are given the prior information that some x_A ($1 < A < n$) has to align to either y_B , or y_C , or y_D ($1 < B < C < D < m$). How do you modify Viterbi to find the most likely alignment under this constraint? (Assume that the prior probability of these three events is uniform.)

(b) You are aligning two proteins $u = u_1\dots u_m$ and $v = v_1\dots v_n$. Assume that the usual 3-state HMM for alignment is a good model for these proteins. Interestingly, your friend Marina has devised a novel experimental technique by which she has determined 100% the “true alignment α^* of the proteins. Instead of the most likely alignment, you are asked to find an alignment α under the following scheme:

(i) (9 points) Marina will pay you \$1 for every letter in u that goes to the correct letter in v . In other words, she will give you \$1 for every aligned pair (u_i, v_j) in α that is also in α^* . Let $G(\alpha, \alpha^*)$ be the number of aligned pairs that the alignments α and α^* have in common (that is, the number of correctly predicted aligned pairs). Describe a way to find the alignment α that maximizes your expected payoff $E_{\alpha^*}[G(\alpha, \alpha^*)]$.

(ii) (8 points) In addition to \$1 for every correct pair (u_i, v_j) , Marina will pay you \$0.5 for every u_i that is correctly gapped between v_j and v_{j+1} , and \$0.5 for every v_j that is correctly gapped between u_i and u_{i+1} . Basically, Marina is paying you \$0.5 for every correctly mapped letter of each sequence. Once again, describe a way to find the alignment α that maximizes your expected payoff.



www.phdcomics.com