Overview

HMMs are powerful statistical modelling tools with widespread applications in bioinformatics. In this HW, you will explore the methodology and theory of several algorithms related to HMM inference.

This assignment is worth a total of 50 points.

Reading

- A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition (Rabiner, 1989)
- Maximum-Likelihood Estimation of Molecular Haplotype Frequencies in a Diploid Population (Excoffier and Slatkin, 1995)
- Identification of CpG islands in DNA sequences using statistically optimal null filters (Kakumani et al., 2012)

Handin

Submit your answers to the following problems as a PDF on Gradescope using your anonymous email address. You may include images of hand-drawn diagrams if necessary, but all written responses must be typed up. Do not include any identifying information on your handin.

P1: The Baum-Welch Algorithm (25 points)

The Baum-Welch algorithm provides an iterative method for “tuning” the parameters of an HMM to infer characteristics of biological sequences. One application of HMMs is detecting CpG islands within genomic sequences. As put by famed computational biologist and bioinformatician Richard Durbin in his seminal book Biological Sequence Analysis (1998):

In the human genome wherever the dinucleotide \( \text{CG} \) occurs (frequently written \( \text{CpG} \) to distinguish it from the \( \text{C-G} \) base pair across the two strands) the \( \text{C} \) nucleotide (cytosine) is typically chemically modified by methylation. There is a relatively high chance of this methyl-\( \text{C} \) mutating into a \( \text{T} \), with the consequence that in general \( \text{CpG} \) dinucleotides are rarer in the genome than would be expected from the independent probabilities of \( \text{C} \) and \( \text{G} \). For biologically important reasons the methylation process is suppressed in short stretches of the genome, such as around the promoters or ‘start’ regions of many genes. In these regions we see many more \( \text{CpG} \) dinucleotides than elsewhere, and in fact more \( \text{C} \) and \( \text{G} \) nucleotides in general. Such regions are called CpG islands [Bird 1987]. They are typically a few hundred to a few thousand bases long.

The presence and location of CpG islands throughout the genome is therefore of great interest to biologists, as it may indicate promoter regions, genes, or chromatin status. In this problem, you will explore how HMMs can be “taught” to “learn” the CpG island status of a given DNA sequence.
The HMM we will use for detecting CpG islands has the following properties:

- 2 hidden states, representing non-CpG island regions (“oceans”) and CpG island regions, respectively (\(S_0 = \text{“ocean”}; S_1 = \text{“island”}\))
- 4 emissions, representing the four DNA bases in alphabetical order (\(v_0 = A, v_1 = C, v_2 = G, v_3 = T\))
- A 2 \(\times\) 2 transition matrix, naively initialized with a greater probability of remaining in the current state rather than transitioning to the other state for both “oceans” and “islands”
- A 2 \(\times\) 4 emission matrix, naively initialized so that the “island” state has a slightly greater probability of emitting the bases \(C\) and \(G\) than the “ocean” state
- A 1 \(\times\) 2 initial state distribution, naively initialized with a greater probability of starting in an “ocean”

We have created a Google Colab notebook (an interactive Python code file shared via Google Drive) with stencil code for the Baum-Welch algorithm here. Your task will be to finish implementing the algorithm and explore its performance on some basic inputs.

To access the Colab notebook, click the link above (you will need to sign in to Google Drive using your Brown email). This will open the notebook in read-only format. To edit the notebook, you can either click File → Save a copy in Drive... or Open in playground → Copy to Drive. You should then be able to edit and save a copy of the stencil code in your own Brown Google Drive account.

Inside the Colab notebook, you can either run individual blocks of code or execute the entire code at once. The notebook will automatically import the following helper functions from our support code file:

- **initialize**: Initializes \(\lambda = (A, B, \pi)\) according to the naive constraints described above (with a small degree of stochasticity)
- **viterbi**: Implements the Viterbi algorithm (a solution to the HMM “Decoding Problem”, covered in CS 181) to compute \(Q^* = \arg \max_{Q} \log[P(O \mid Q, \lambda)]\)
- **compute_logP**: Computes the total log probability of observing the sequence (\(\log[P(O \mid \lambda)]\))
- **print_results**: Prints the total log probability of observing the sequence (\(\log[P(O \mid \lambda)]\)), the optimal state sequence (\(Q^*\)), and the probability of the optimal state sequence (\(\log[P(O \mid Q^*)]\)) for each iteration of the algorithm

The function **run_baum_welch** implements the algorithm, which runs until convergence is achieved, using the helper functions to produce interpretable results at each iteration. **Do not modify this function!**

Your task will be to implement the following seven key functions, upon which **run_baum_welch** relies:

- **calc_alpha**: Calculate the forward variable (\(\alpha\))
- **calc_beta**: Calculate the backward variable (\(\beta\))
- **calc_xi**: Calculate the \(\xi\) variable
- **calc_gamma**: Calculate the \(\gamma\) variable
- **update_A**: Update the transition matrix (\(A\))
- **update_B**: Update the emission matrix (\(B\))
- **update_pi**: Update the initial state distribution (\(\pi\))

You may find the posted lecture notes from CH4 useful in translating the formulas and relationships between these variables into code, as well as Rabiner’s “tutorial” on HMMs (provided in the Readings above). Each function should be relatively straightforward to implement if you follow the notation correctly. Note that you may need to store probabilities as log probabilities in order to avoid underflow.

When you have finished implementing the seven key functions above, you should be able to actually run the Baum-Welch algorithm on the two example sequences provided in the notebook!
**Note:** We recognize that not all students prefer to use Python, and that some students may not have had prior experience working with the NumPy library or Google Colab notebooks. Please don’t hesitate to reach out to the TAs if you have questions about syntax or code structure. The intention of this HW problem is to provide you with an opportunity to see the Baum-Welch algorithm in action, not to reach the level of coding complexity required to solve the PR problems.

Submit the **text of your code** for the seven key functions described above in your handin (we will not be running your code, but we will be checking your implementation of each function for correctness).

Start by running the Baum-Welch algorithm on the first sequence provided. This is a simple example DNA sequence containing a putative CpG island. Answer the following questions (note that Q1 can be answered before any of the seven key functions have been implemented):

1. Run the algorithm 15 – 20 times and examine the **first line of output only** (representing $Q^*$ for the naive $\lambda$ prior to any iterative tuning). Since the **initialize** helper function is not deterministic, each run of the algorithm will start from a different initial $\lambda = (A, B, \pi)$. What patterns do you observe in the initial $Q^*$ sequence? (Note that if you have not yet implemented any of the seven key functions, the total log probability of the sequence ($\log[P(\mathcal{O} \mid \lambda)]$) will erroneously be printed as `-inf`. You can ignore this.)

2. Once all seven key functions have been implemented, run the algorithm 15 – 20 times and examine the results. You should be able to get a sense of how the total log probability of the sequence ($\log[P(\mathcal{O} \mid \lambda)]$) converges to a local maximum, as well as how $Q^*$ and the Viterbi probability ($\log P(\mathcal{O} \mid Q^*)$) change as the algorithm “tunes" $\lambda$. What kinds of patterns in the final $Q^*$ do you observe? Where do you think the CpG island is located within this sequence? (You may provide representative outputs if useful.)

3. You should be able to observe a variety of different $Q^*$ sequences across multiple runs of the algorithm. Comment on the range of inferred “best states”. What do these results suggest about the overall strengths and limitations of the Baum-Welch algorithm?

4. What kinds of patterns do you observe in the final tuned $\lambda^* = (A^*, B^*, \pi^*)$? Interpret the relative values of $A^*$, $B^*$, and $\pi^*$. What has the HMM “learned”?

Now run the Baum-Welch algorithm on the second sequence provided. This is a 500-bp subsequence of the **TP53** gene, which in humans codes for the p53 protein (nicknamed the “guardian angel of the genome” because it is the most frequently-mutated gene in human cancers). According to the UCSC Genome Browser, this sequence contains a 213-bp CpG island, which is located at the beginning of the **TP53** gene.

5. Run the algorithm 5 – 10 times and comment on the results you observe. Can you verify the location of the CpG island?

6. Why might inference of $Q^*$ from this sequence not be as straightforward or unambiguous compared to the first sequence?

The HMM we are tuning in this problem is fairly simple.

7. What is one drawback of using this HMM to “learn” the position of CpG islands within a sequence?

   **Hint:** You may find Durbin’s excerpt above helpful.

8. What are some improvements that could be made to combat these limitations? Present a structure for an alternative HMM for CpG island detection (define its hidden states and emissions, and the dimensions of its transition, emission, and initial state distribution matrices). What kinds of general patterns might you expect to see in its final tuned transition and emission matrices ($A^*$ and $B^*$)?

   **Hint:** You may find the paper by Kakumani et al. (provided in the Readings above) helpful.
The Baum-Welch algorithm is a special case of the more general Expectation-Maximization (EM) algorithm. In this problem, you will work through the mathematical theory that makes the EM algorithm such a powerful optimization tool.

Note: The following proofs are mathematically challenging. Don’t hesitate to ask questions!

A function $f$ is defined as convex on the domain $[a, b]$ if it satisfies the inequality

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

for all $x_1, x_2 \in [a, b]$ and $\lambda \in [0, 1]$. Intuitively, this condition means that the function will lie on or below the secant line joining $f(a)$ and $f(b)$ over the region $a \leq x \leq b$.

1. Let $f$ be a convex function and let $X$ be a random variable. Use the definition of convexity above to prove the following probability theorem (Jensen’s inequality):

$$f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$$

Hint: You may find induction useful.

Importantly, both of these inequalities can be inverted if $f$ is concave instead of convex.

The objective of the EM algorithm is to maximize the log-likelihood of the data. We will now examine how each iteration of the algorithm achieves this goal.

2. Let $L(\theta) = \log[P(x | \theta)]$ be the likelihood of the data sample $x = \{x_1, \ldots, x_n\}$ given some parameters $\theta$, and let $z$ be a set of “hidden” (or unknown) data. Show that for each iteration of the algorithm,

$$L(\theta') - L(\theta) = \log \left[ \sum_{z_i} P(x | z_i, \theta') P(z_i | \theta') \right] - \log[P(x | \theta)]$$

The process of conditioning the data sample $x$ on a set of “hidden” data $z$ is useful in maximizing the difference above over each iteration, especially when $L(\theta)$ itself is not easily maximized.

Bonus (5 points): Show that the introduction of the “hidden” data $z$ provides an alternative means of maximizing the likelihood above, using the following property derived from Jensen’s inequality and the concavity of $\log[x]$ (for $\sum_{i=1}^{n} \lambda_i = 1$, $\lambda_i \geq 0$):

$$\log \left[ \sum_{i=1}^{n} \lambda_i y_i \right] \geq \sum_{i=1}^{n} \lambda_i \log[y_i]$$

Hint: Consider $\lambda_i = P(z_i | x, \theta)$.

**P3: Reading Questions (10 points)**

Read through the paper by Excoffier and Slatkin linked above and answer the following questions:

1. What are the two assumptions required for the EM algorithm to be applied to haplotype phasing?
2. Given $k$ genotypes with $n$ polymorphic sites, what is the approximate runtime of this algorithm?
3. What are some difficulties that arise in initializing this algorithm with uniform probabilities?
4. What are some changes that could be made to this algorithm to improve its runtime (consider how the information which is “known” after each iteration could be used to shrink the search space)?