My primary attempt to reduce both runtime, and mathematical complexity, I structured the code in such a way that as much of the math was done in the __init__ method, and then looking them up while the primary tagging process executes. I came to the dynamic programming solution that I used after a bit of research on the Viterbi algorithm, I believe the textbook describes it as dynamic. My first attempt I used a straightforward recursion algorithm that would attempt every single combination of part of speech tags in the sentence. Due to some other problems that I ended up scrapping instead of solving, this took on the order of 45 minutes to run.

0 probabilities are handled in an aggressively simple manner. The process checks to see if a word has a probability to be a tag which is greater than 0. When it finds a tag which is never associated with a word it assumes that those probabilities will be 0, and does not consider them. For example the word “survive” is never used as a pronoun, and it would be illogical to assume that it might be. The only error I encountered this way was handling the probability related to the start of the sentence. I believe these to have been cases where a word was never seen to be the start of a sentence. I felt this to be a symptom of some of the messier parts of language, which is to say that english strongly discourages starting a sentence with certain parts of speech.

For words which had appeared in the test, but not training data. The code generates an entry in the training data for the found entry, and assigns it probabilities equal to the average of the whole model.

In both the english and french tests, the HMM system produces results with a higher accuracy for both distinct and ambiguous tags than the baseline systems. 94.8% compared to 96.9% when using the HMM model in english over the baseline model, which is slightly below a 50% decrease in words improperly tagged. The french corpus saw a roughly similar increase from 84.4% to 87.6%, but that would mean that there is a much smaller difference between the amount of words improperly tagged.

There are 38 english words in the test corpus which had been improperly identified. I would assume that given the structure of the program, they would be clustered together in sentences with each other, which is to say that one mistake would cascade down into several in the sentence. An improvement which would not be overly expensive for the program’s runtime would be to track not just the highest probability backpoint as detailed in the viterbi algorithm, but to track the highest 2 or 3 back pointers, and run through the program and select a second or third back pointer on second or third examinations of sentence structure (if available) and then add those results to the pool of candidates being considered.