Problem Set 3: HMMs

The University of Texas at Austin
CS395T/EE381V: Spoken Language Technologies
Fall 2022
Instructor: David Harwath

Assignment Date: September 22, 2022

Due Date: October 4, 2022

This problem set consists of 2 problems (each with multiple parts), totaling 100 points.

You will explore:

1. Working through several written problems on HMMs **(25 points)**
2. Coding several HMM algorithms and implementing a fully-functional isolated word recognizer **(75 points)**

To turn in: You should submit a single .pdf writeup to Canvas, containing all of your deliverables for all parts of the problem set. For the written parts of the assignment, you must show your work. For the coding parts of the assignment, you must turn in all of the plots and code fragments that are specified as deliverables in the corresponding exercises.

Collaboration Policy: You may collaborate in groups of no larger than 3 students, and each student must turn in their individual writeup written in their own words. **You must clearly indicate all of your collaborators in the title block of your writeup.**

Late policy: Sometimes we have bad days, bad weeks, and bad semesters. In an effort to accommodate any unexpected, unfortunate personal crisis, I have built "time banks" into our course. You do not have to utilize this policy, but if you find yourself struggling with unexpected personal events, I encourage you to e-mail me as soon as possible to notify me that you are using our grace policy. You may use this policy one of two ways (please choose, and let me know): You may have a two-day grace period for one assignment, OR You may have 2 one-day extensions for two different assignments. Late assignments will otherwise not be accepted.

Exercise 1 (25 pts)

Consider the 4-state, discrete HMM defined in Figure 1:

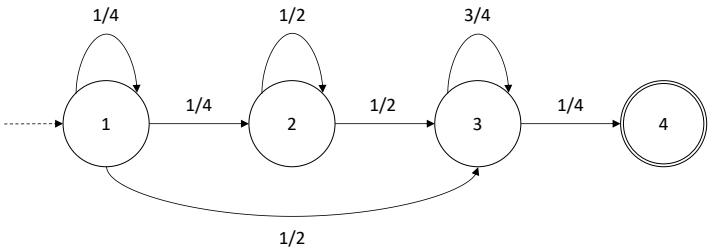

Figure 1: A 4-state HMM

The symbol emission distribution for the HMM is defined in Table 1:

	State 1	State 2	State 3
P(H)	1/2	3/4	1/4
P(T)	1/2	1/4	3/4

Table 1: HMM symbol emission distribution

You can also assume that $\pi_1 = 1$, and $\pi_s = 0$ for $s = 2, 3, 4$. Finally, we will constrain state 4 to be the only accepting state; in other words, **all state sequences must terminate in state 4, which is also a non-emitting state.** Let λ represent all of the above HMM parameters.

Assume that the symbol sequence $o_1 o_2 o_3 = HTT$ was observed, and answer the following questions.

Q1: List all of the possible hidden state sequences that could have produced the observation sequence HTT

Q2 What is the most likely hidden state sequence, conditioned on the observation sequence?

Q3 What is $P(HTT|\lambda)$?

Q4 What is the probability of being in state 1 at time $t = 2$, conditioned on the observation sequence?

Exercise 2 (75 pts)

In this exercise, you will implement an HMM-based isolated word recognizer. I've provided the front-end feature extraction code as well as the already-trained acoustic model for you, and your job will be to finish the HMM implementation by writing the code for:

1. Sequence scoring using the Forward Algorithm
2. State-level decoding using the Viterbi algorithm
3. Maximum likelihood re-estimation of the transition matrix, using Viterbi training.

In theory, you can use your front-end feature extractor from Lab 1 (with some minor modifications) and your acoustic model from Lab 2 instead of the ones I have provided - thus, by putting together all 3 labs so far you'll have implemented a complete word recognizer, from waveform to recognition result.

We have provided the following files for you to use:

1. `lab3.ipynb` : The Jupyter notebook containing the starter code
2. `acoustic_model.pt` : The already-trained acoustic model you can use to compute frame-level phone likelihoods
3. `lab3_phone_labels.npz` : The mapping from (integer) phonetic class indices to (string) phonetic labels
4. `fee.wav`, `pea.wav`, `rock.wav`, `burt.wav`, `see.wav`, `she.wav` : 6 waveform files that you can use to test your recognizer

2.1 Computing class likelihoods

We have provided two functions to you:

- `load_audio_to_melspec_tensor`: This function takes as input the path to a .wav file, and returns a PyTorch tensor of size (N_frames, 40) representing the log Mel spectrogram of the .wav file, computed over 40 Mel filters.
- `compute_phone_likelihoods`: This function takes as input a deep neural network acoustic model, as well as a PyTorch tensor containing the log Mel spectrogram computed with `load_audio_to_melspec_tensor`. The output will be a numpy array of shape (N_timesteps, 48), where the first dimension indexes acoustic frames and the second dimension indexes the **log likelihoods** across the 48 different phoneme classes. In practice, when coding probablistic models we typically manipulate probabilities and likelihoods in the log domain (i.e. $log(p(x))$ instead of $p(x)$) in order to avoid numerical underflow when multiplying probabilities together.

Using these two functions, you can compute the frame-level phoneme likelihoods for all 6 of the waveforms we have provided, which will serve as the inputs to the HMM functions you will implement. We've provided a skeleton HMM class to you, as well as several lines of code that instantiate an HMM object for each of the 6 words we will try to recognize. You will need to fill in the three functions outlined below.

2.2 Implementing the Forward Algorithm

The first function you should implement is `forward_score`, which should compute $p(O|\lambda)$, where $X = o_1, o_2, ...o_N$ are the acoustic frames representing a speech waveform, and λ are the acoustic model and HMM parameters.

We'll be using a slightly modified version of the Forward algorithm from the one presented in class:

1. Initialization: $\alpha_1(i) = b_i(o_1)\pi_i$
2. Induction: $\alpha_{t+1}(i) = [\sum_{j=1}^{N} \alpha_t(j)a_{ji}]b_i(o_{t+1})$
3. Termination: $p(o_1, ..., o_T, q_T = s_N|\lambda) = \alpha_T(N)$
   (this is where we make the minor modification - rather than summing α over all the states, we want to constrain only the final state to be an accepting state, to reflect the entire word being spoken instead of only part of it).

where:

- $\alpha_t(i)$ represents the joint likelihood of having seen the observations up to time t $o_1, o_2, ..., o_t$ *and* being in state i at time t.
- $b_i(o_t)$ is the likelihood of emitting observation o_t in state i
- a_{ij} is the probability of transitioning from state i to state j
- π_i is the probability of being in state i at time $t = 1$
- $q_T = s_N$ indicates we are in state s_N at time T

Connecting the provided skeleton code to the above notation, the `compute_phone_likelihoods` function will return a 2-dimensional array `L` where `L[t,k]` = $\log p(o_t|c_k)$, where c_k is the k^{th} phonetic class (out of 48 returned by the acoustic model). The `MyHMM` class is parameterized using the following instance variables:

- `N_states`: An integer value representing the number of total states in the HMM
- `pi`: A 1-dimensional array of size `N_states`, where `pi[i]` = $\log \pi_i$
- `A`: A 2-dimensional array of size `N_states` by `N_states` representing the transition matrix, where `A[i,j]` = $\log a_{ij}$
- `labels`: A 1-dimensional array of size `N_states`, where `labels[i]` indicates the phonetic identity (class index) of state i. This can be used to retrieve the phonetic label of the state via `phone_labels[labels[i]]`, or to retrieve the observation likelihood $b_i(o_t)$ via `L[t, labels[i]]`

Note that the `L` matrix of phonetic likelihoods reflects the observation likelihoods of the feature frame conditioned on all of the 48 possible classes, but you won't actually need to use this full matrix in any individual word HMM - you only need to select the columns that correspond to the phones actually used in a particular HMM. You can use the `lab3_phone_labels.npz` file to identify which columns correspond to which phone labels.

In order to compute the Forward algorithm in a numerically stable way, you'll want to re-write the algorithm to use log probabilities/likelihoods rather than raw probabilities/likelihoods.

You will want to use the `logsumexp` function from the `scipy.special` library to compute the summation in the Induction step.

The output of your forward algorithm should be a log likelihood, representing $\log p(o_1, \ldots o_T, q_T = S_N|\lambda)$ for a given input waveform and the current setting of the HMM parameters.

2.3 Implementing the Viterbi Decoding Algorithm

Recall the Viterbi algorithm from lecture:

1. Initialization: $\delta_1(i) = b_i(o_1)\pi_i, \quad \Psi_1(i) = 0$
2. Induction: $\delta_t(i) = [\max_j \delta_{t-1}(j)a_{ji}]b_i(o_t), \quad \Psi_t(i) = \arg\max_j \delta_{t-1}(j)a_{ji}$
3. Termination: $q_T^\star = \arg\max_i \delta_T(i)$
4. Backtrace: $q_t^\star = \Psi_{t+1}(q_{t+1}^\star)$

The Viterbi algorithm is very similar to the Forward algorithm, with two notable differences. First, in the induction step, we replace the summation with a max. Second, we keep a backtrace matrix Ψ that we can use to remember the best state at time $t-1$ that transitioned to some given state at time t.

Implement the Viterbi decoding algorithm in the `MyHMM` class, and have it return the best hidden state sequence for the input observation sequence.

2.4 Implementing Viterbi Training

Finally, use the Viterbi decoding algorithm you just implemented as a helper function for `viterbi_transition_update`, which should perform a maximum likelihood update (using only the Viterbi 1-best path - this is called Viterbi training) of the HMM's transition matrix, given an input observation sequence.

Compared to full Baum-Welch updates, Viterbi training is quite straightforward. The transition matrix update equation is

$$A[i,j] := \frac{\text{\#of times a transition was made from state i to state j}}{\text{\#of times the model made a transition out of state i}}$$

These probabilities will be safe to compute as raw probabilities, but once you actually assign them to the `A` matrix you will want to take the logarithm. To avoid taking the log of 0, you will want to add a very small ε value to each probability before taking the log - take a look at the `__init__()` function in the `MyHMM` class to see how this is done.

Now that you've finished coding up your HMM class, use it to answer the questions below:

Likelihood computation. Compute the log likelihoods for all 6 waveforms with all 6 of the word HMMs. Organize your results into a 6x6 table. The recognition result for a given word in our case is simply the identity of the HMM that achieved the highest likelihood for the observation sequence. Which of the words were correctly recognized?

Optimal state sequence. Print the optimal hidden state sequence for the word "rock" using the HMM representing the word "rock" and paste it into your writeup.

Viterbi Update. Perform a Viterbi update of the transition matrix for the HMM representing the word "rock," using the `rock.wav` file. Next, compute the likelihood of `rock.wav` with the

updated HMM. What is the new likelihood? Did it go up or down? Print the new transition matrix (you can exponentiate the log probabilities before you print them to make it more readable). How did it change?