11-751/18-781 Speech Recognition and Understanding (Fall 2023)
Coding Assignment 2

OUT: September 27th, 3:30 PM ET
DUE: October 13th, 11:59 PM ET

Instructor: Prof. Shinji Watanabe
TAs: Xuankai Chang, Brian Yan, Yifan Peng

Collaboration Policy

Assignments must be completed individually. You are allowed to discuss the homework assignment with other students and collaborate by discussing the problems at a conceptual level. However, your answers to the questions and any code you submit must be entirely your own. If you do collaborate with other students (i.e. discussing how to attack one of the programming problems at a conceptual level), you must report these collaborations. Your grade for Coding Assignment 2 will be reduced if it is determined that any part of your submission is not your individual work.

Collaboration without full disclosure will be handled in compliance with CMU Policy on Cheating and Plagiarism: https://www.cmu.edu/policies/student-and-student-life/academic-integrity.html

Late Day Policy

You have a total of **5 late days (i.e., 120 hours)** that you can use over the semester for the assignments. If you do need a one-time extension due to special circumstances, please contact the instructor (Shinji Watanabe) via Piazza.

Programming Notes for Coding Assignment 2

1. The Gradescope testing environment uses Python 3, specifically Python 3.8 with numpy(1.24), scipy(1.10), and scikit_learn(1.3.1). Make sure your code is compatible with the same version.
2. Other packages not mentioned above (`numpy`, `scipy`, `scikit-learn`) are **not allowed** by the auto-grader, so please do not use any others. Actually, you will not need them.
3. You will not be able to see printed outputs or debugging statements when you submit to Gradescope.

Problem 1

Keyword Classification (5 pts)

Gradescope Assignment Link: https://www.gradescope.com/courses/564396/assignments/3392268

Hidden Markov Model-Gaussian Mixture Models are used for Acoustic Modelling in classical Speech Recognition, with the HMM representing different hidden context dependent phone states, and the GMM modelling the emission probabilities of the HMM. In this assignment, you will solve a simpler version of the problem by building a HMM-GMM model for speech based keyword classification. Acoustic modelling is preceded by feature extraction, which has been done for you in this problem. The focus is on building the basic diagonal Gaussian model, which is then used to build a Gaussian Mixture Model, which is used with the HMM for acoustic modelling.

Diagonal Gaussian Distribution (1 pt)

The basic element in this assignment is the multivariate Gaussian distribution with diagonal covariance matrix. Recall the probability distribution of a D-dimensional standard Gaussian distribution is defined as follows:

$$\mathcal{N}(\mathbf{o}|\mu, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{o}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{o}-\mu)\right) \tag{1}$$

where $\mathbf{o} \in \mathbb{R}^D$ is the observation , $\mu \in \mathbb{R}^D$ is the mean, $\mathbf{\Sigma} \in \mathbb{R}^{D\times D}$ is the covariance matrix.

As we are dealing with the Diagonal Gaussian distribution, the formula can be simplified by using the diagonal covariance $\mathbf{r} = [\mathbf{\Sigma}_{11}, \ldots, \mathbf{\Sigma}_{DD}]$.

$$\mathcal{N}(\mathbf{o}|\mu, \mathbf{r}) = \prod_{d=1}^{D} \mathcal{N}(o_d|\mu_d, r_d) \tag{2}$$

$$= \prod_{d=1}^{D} \left(\frac{1}{\sqrt{2\pi r_d}} \exp\left(-\frac{(o_d-\mu_d)^2}{2r_d}\right)\right) \tag{3}$$

Using Equations 3, the log pdf of the diagonal Gaussian can be derived as follows

$$\log(\mathcal{N}(\mathbf{o}|\mu, \mathbf{\Sigma})) = -\frac{D}{2}\log(2\pi) - \frac{1}{2}\sum_{d=1}^{D}\log(r_d) - \frac{1}{2}\sum_{d=1}^{D}\frac{(o_d-\mu_d)^2}{r_d} \tag{4}$$

Gaussian Mixture Model (GMM) (1 pt)

Let's assume that the Gaussian Mixture Model (GMM) has K components (Gaussian distributions). We are given the observations $\mathbf{O} = \left(\mathbf{o}_t \in \mathbb{R}^D | t = 1, \ldots, T\right)$ and latent variables $V = (v_t \in \{1, \ldots, K\} | t = 1, \ldots, T)$. The likelihood of GMM for $\mathbf{o}_t$ can be written as a linear combination of Gaussian distributions as follows:

$$p(\mathbf{o}_t|\Theta) = \sum_{k=1}^{K} p(\mathbf{o}_t|v_t = k, \Theta) p(v_t = k|\Theta) \tag{5}$$

$$= \sum_{k=1}^{K} \omega_k \mathcal{N}(\mathbf{o}_t|\mu_k, \mathbf{r}_k) \tag{6}$$

where $\omega \in [0, 1]^K$ is the weight of each component. To compute the likelihood on the full observations, it can be written as follows:

$$p(\mathbf{O}|\Theta) = \sum_{V} p(\mathbf{O}, V|\Theta) \tag{7}$$

$$= \prod_{t=1}^{T} \sum_{k=1}^{K} \omega_k \mathcal{N}(\mathbf{o}_t|\mu_k, \mathbf{r}_k) \tag{8}$$

In traditional speech recognition models, each GMM typically represents the density distribution of a single state for a mono-phone or tri-phone. To reduce the number of total parameters, one GMM distribution could even be shared by multiple phonemes.

It is well-known that GMM with more than two components (i.e. $K > 1$) does not have a global optimum using MLE estimation- we can improve the total likelihood by fitting one component to one point and shrinking its variance to 0. Therefore we would like to find a good local optimum for MLE with the Expectation Maximization (EM) algorithm. In this assignment, we ask you to **implement both the E step and M step for the Gaussian Mixture Model**.

Expectation Step In the E step, we would like to compute the responsibility parameter $\gamma_k(t)$ for the k-th Gaussian component at the t-th time step, where $k \in [1, \ldots, \mathrm{K}]$ and $t \in [1, \ldots, \mathrm{T}]$. It can be estimated with current set of GMM parameters as follows:

$$\gamma_k(t) = p(v_t = k|\mathbf{o}_t, \Theta') = \frac{\omega_k \mathcal{N}(\mathbf{o}_t|\mu'_k, \mathbf{\Sigma}'_k)}{\sum_{k'=1}^{K} \omega_{k'} \mathcal{N}(\mathbf{o}_t|\mu'_{k'}, \mathbf{\Sigma}'_{k'})} \tag{9}$$

Maximization Step In the M step, we update the GMM parameters (weights,means and variances) based on the current responsibilities.

$$\hat{\omega}_k = \frac{\sum_{t=1}^{T} \gamma_k(t)}{\sum_{t=1}^{T} \sum_{k'=1}^{K} \gamma_{k'}(t)} \tag{10}$$

$$\hat{\mu}_{k,d} = \frac{\sum_{t=1}^{T} \gamma_k(t) o_{t,d}}{\sum_{t=1}^{T} \gamma_k(t)} \tag{11}$$

$$\hat{\Sigma}_{k,d} = \frac{\sum_{t=1}^{T} \gamma_k(t)(o_{t,d} - \hat{\mu}_{k,d})^2}{\sum_{t=1}^{T} \gamma_k(t)} \tag{12}$$

By iteratively applying the E and M steps, we might get a good local optimum of parameters. However, as we mentioned above, it is possible that a Gaussian component fits a single point by shrinking its variance

towards 0. To prevent such a case, you might want to restrict your variance to not be too small, and reset the mean of that component to some other places.

Hidden Markov Model (HMM) (3 pts)

Hidden Markov Model is a generative model used in acoustic modeling. In this assignment, we use a HMM to represent *yes* and *no*. The HMM can be connected with GMM as illustrated in Figure.1. Each red cycle is a hidden state in HMM, and typically corresponds to one phone (or triphone). Each state has two types of transitions: a recursive self-transition (model remains in the same state), or a forward transition (moves forward to the next state). The recursion means that the phoneme is longer than 1 frame.

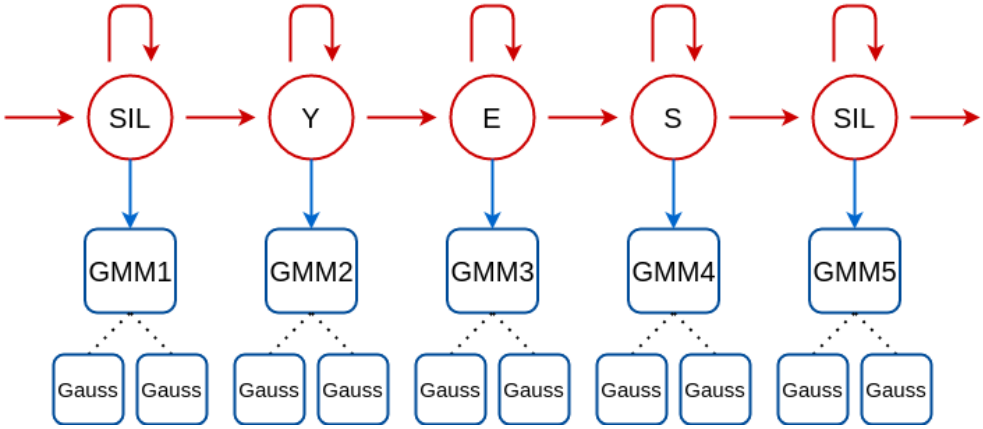

Figure 1: The linear topology of GMM-HMM acoustic modeling. There are 5 states in the HMM where the first state and last state represents the silence region in the audio, each of the other states represents one phoneme. Every state has a GMM as the emission model, which typically has multiple Gaussian distributions.

Transition Model The state transition probability from state i to j can be written as $p(s_t = j|s_{t-1} = i) = a_{i,j}, i, j \in \{1, \ldots, J\}$, where J is the number of state (there are 5 states in this figure). Guess what the initial state probability $p(s_1)$ looks like. As shown in the Figure, there are only two probable transitions for each state, therefore $a_{i,i} > 0, a_{i,i+1} > 0$ and $a_{i,i} + a_{i,i+1} = 1$.

Emission Model The emission model is typically a probabilistic model. In our architecture, each state is associated with a GMM as illustrated. The GMM is used to model the feature density distribution of the phonemes represented as states. Inside each GMM, there are usually multiple Gaussian components to represent variety from different speakers and recording environments.

In this assignment, we would like to train an acoustic model with the following steps:

Step 1: Equal alignment Initially, when the HMM-GMM model has not been initialized, we need a default alignment over the HMM states to initialize the GMM parameters. For each sample observation, we equally align its frames into each of the HMM state. Suppose that the observation has 100 frames, then the first 20 frames are assigned to first state SIL, the next 20 frames are assigned to Y and so on. We apply this equal alignment to all samples in our training set.

Step 2: Update HMM and GMM (M step) By aligning all of our samples in the training set, we have good statistics to update the GMM and HMM. The transition probability $a_{i,i}$ in HMM can be updated as follows

$$\hat{a}_{i,i} = \frac{\sum_{t=2}^{T} \xi_{t-1}(i,i)}{\sum_{j=1}^{J} \sum_{t=2}^{T} \xi_{t-1}(i,j)} \tag{13}$$

where $\xi_t(i,j)$ is the probability of transition from i-th state to j-th state in t-th sample. As the sum of forward probability and recursion probability is 1.0, the new forward transition probability to the next state $i+1$ is obtained by

$$\hat{a}_{i,i+1} = 1 - \hat{a}_{i,i} \tag{14}$$

Note that this is an approximation to estimate transition probability, but it works well in practice. The emission model is updated by its own EM steps, which was discussed in the GMM section above. Each GMM should be updated based on the frames assigned to its state by alignment.

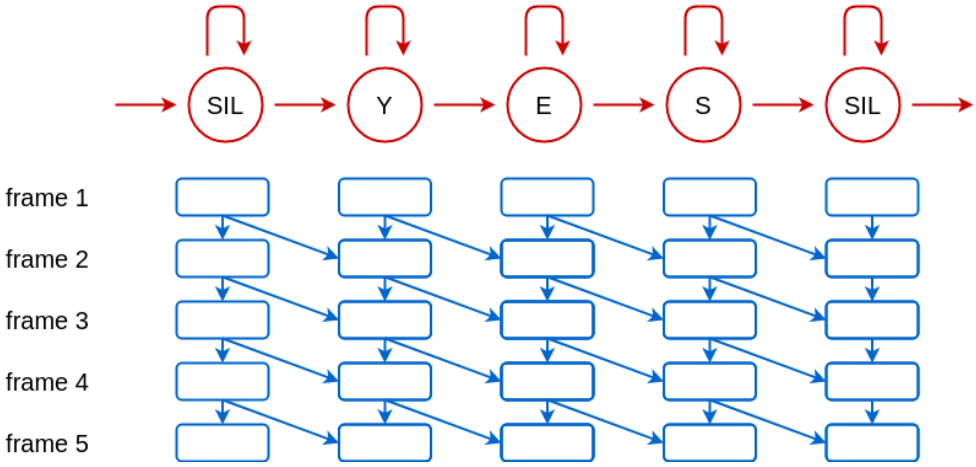

Figure 2: The Viterbi search graph.

Step 3: Align with Viterbi align (E step) In the E-step, the parameters of HMM and GMM are fixed and frames of each observation are re-aligned. The alignment can be implemented as a Viterbi search illustrated in Figure.2 where each frame has a score for each state. The cell at the t-th frame and j-th state shows the log probability of most probable path reaching that frame and state.

The score $\delta_t(j)$ at t, j-th cell can be computed with

$$\delta_t(j) = \log p(\mathbf{o}_t|s_t = j) + \max\{\delta_{t-1}(j) + \log(a_{j,j}),\ \delta_{t-1}(j-1) + \log(a_{j-1,j})\} \tag{15}$$

where $p(o_t|s_t = j)$ is the j-th state' GMM probability for t-th frame in this observation. By computing the score for each reachable cell, we can obtain the Viterbi alignment by backtracking from the last state and last frame. By repeating the alignment process for every observation, we could again extract a new set of frames corresponding to each state, which would be further used to optimize the model as in Step 2.

Step 3.1 (Optional): Forward-Backward, a.k.a. Baum-Welch algorithm (E step) In the E-step, the parameters of HMM and GMM are fixed and frames of each observation are re-aligned. Instead of using Viterbi align, you may also use the Forward-Backward (Baum-Welch) algorithm to updated. This step is optional with bonus. You can get familiar with the algorithm which may be used in the future assignments.

Step 4: Repeated EM Optimization Repeat Step 2 and Step 3 until convergence or for a fixed number of iterations.

Step 5: Keyword Classification with the Forward algorithm After training both HMM and GMM, we can use the model to estimate the probability for keywords. For this, we will use the Forward Algorithm rather than Viterbi search. The forward algorithm probability α can be computed in a similar way to

Viterbi search, with one difference: instead of finding the maximum probability path, we sum up scores for all available paths to obtain the marginalized probability

$$\alpha_t(j) = \log p(\mathbf{o}_t|s_t = j) + \text{logsumexp}(\alpha_{t-1}(j) + \log(a_{j,j}), \alpha_{t-1}(j-1) + \log(a_{j-1,j})) \tag{16}$$

Task: Write python code to implement a Diagonal Gaussian model, GMM and HMM. Use the provided template in the handout, and add your code to the template as directed. You are not allowed to use any library other than *numpy*. The only exception is *logsumexp* from *scipy*. Check Tips for details.

Helper code- hmm_kwc directory in the handout.

1. gauss.py - the Diagonal Gaussian model template, do not change the class or any interface names. Implement the *fit* and *logpdf* methods.
2. gmm.py - the GMM template. Implement the *E_ step* and *M_ step* methods. You can change other code as well, but do not change the method names or their interfaces. In the *E_ step*, you should compute the responsibility z_{nk} for each sample and Gaussian component, in the *M_ step*, use the responsibility score to update your GMM with new (μ, Σ, π) parameters.
3. hmm.py - the HMM template with Viterbi Search. Implement the *align*, *update* and *logpdf* methods. In *align* method, we expect you to implement Viterbi search to find a good alignment. Refer to *align_ equally* method to see the alignment sample you should generate. In the *update* function, we would use the accumulated statistics to update both HMM and GMM. In particular, you should update the transition probability in HMM and call the *fit* interface you implemented in *gmm.py*. In *logpdf*, we expect you to compute the marginalized probability for the keyword. You can use the similar procedure in the Viterbi search. Feel free to update other methods as well.
4. hmm_forward_backward.py - the HMM template with forward-backward algorithm (**Optional**). To be updated.
5. model.py - This is the classification model, it would create multiple HMM instances for each keyword and fit the training set. We have already implemented all the necessary features in this class, but feel free to change anything in this class.
6. submit.py - This is a file to load the training set, train the acoustic model and generate outputs for the test set. Feel free to change anything in this class as well. In particular, you might want to find the best configuration (i.e.: state number and component number) of your HMM model. However, the number of components should not be less than 2 because we want a GMM model not a single Gaussian.

Dataset

1. train.txt - This is the text file containing labels of training data. The first field is the name of the feature file, the second file is the label field where 1 and 0 means *yes* and *no* respectively.
2. train directory: This is the directory containing all the training features. Each file contains the MFCC feature extracted from one audio clip of either *yes* or *no*
3. test.txt - This is a sample submission file. You should train your model and generate a file in the same format: the first field is the file name, the second field is either 0 or 1. Note that the first field should be sorted as this sample submission.
4. test directory: This is the directory containing all the test files. There are 2000 files in total.

Your Gradescope submission: Submit the following files in one zip file named *your-andrewid.zip*

1. gauss.py - We will test your implementation by first calling *fit*, and then *logpdf*. (1 pt)
2. gmm.py - We will test your implementation by first calling *fit*, and then *logpdf*. Although we ask for Viterbi search in the Step 3 (E-step), if the forward-backward is correctly implemented, you will get 1 bonus point (extra credit) (1 pt (+1 pt).
3. hmm.py - This will be graded based on your *test.txt* performance. We will NOT test this class directly, but will manually check your implementation.
4. model.py - Submit this file if you made any change to this template. We will NOT grade this file.
5. submit.py - Submit this file if you made any change to this template. We will NOT grade this file.
6. test.txt - This should contain 2000 lines in the format we mentioned above. This will be graded automatically. Your score will be graded linearly based on your performance between 50 % to 90 % accuracy. 90 % accuracy will receive the full score. You will get 1 bonus point (extra credit) if you achieve more than 95 % accuracy. (3 pts (+1 pt))
7. others - Submit any other files if necessary.

Gradescope Leaderboard: For this assignment, we will use a Gradescope Leaderboard for which you will be asked to provide a Leaderboard name for your submission. Use a name that does not identify you on the Leaderboard. The Leaderboard will allow you to see your test accuracy score and refine your model for future submissions.

Tips

We summarize some tips you might find useful in this problem. Please look over these tips and make sure you are following them before approaching us with code related questions.

1. As most of the alignment and inference depend on log probability, you might want to use *logsumexp* function from *scipy.misc* when summing up multiple log based probabilities. This is the only function you are allowed to use from *scipy* package.
2. You might want to verify your Gaussian model and GMM implementation by comparing its outputs with the results from *GaussianMixture* from *sklearn.mixture*. For the single Gaussian model, set the component number to 1 in *GaussianMixture*.
3. The default configuration (i.e: 10 state size and 3 component size) should give you reasonable results. We got 93.6% accuracy on the training set and 92.1% accuracy on the testing set. Feel free to change those parameters.

Here are some debugging tips. Hope these would be helpful to debug your models. Be patient and diligent in debugging.

- gauss.py
    1. std is the square root of variance.
    2. **Initialization is important.** You may use a kmeans to intialize the GMM. However, if your kmeans initialization end up with some unbalanced assignments, it will affect your GMM results. For example, if you get some initial assignment like [1,2,3,3,3,3] for 3 cluster, then it probably will make subsequent EM wrong. because only the first point is fitting to the first cluster, collapsing its std to 0, and some later NaN issues. You should be careful when the initial assignments are not balanced. kmeans++ usually can avoid this issue (not always though). If std is still becoming very small, try setting a small threshold: std = max(std, 10^{-3}).

3. To debug your model, use sklearn.GaussianMixture and set the component to 1, your logpdf should match almost exactly with the score from the sklearn model.
4. To debug your model, use sklearn.GaussianMixture and make your score near its score (we have a large tolerance range to allow some precision mismatch for this problem).
5. Use low-dimension simple test cases, e.g: 6 point [1,2,10,11,100,101] with 3 cluster, can you get same logpdf for each point after fitting? if not, something is wrong.
6. When computing the product for probability, it is summation in log domain: $\log + \log$.

- hmm.py
  1. Again be careful about log space. Please don't change prob graph[0][0] = 1.0 (it is equivalent to zero probability because we are doing it in the log domain). Some students forget taking log or exp in the right place.
  2. When the frame is 0, only the first state has 1.0 probability mass, other states should not have any mass.
  3. Don't simply taking argmax to extract alignments from graph, this is not enough. You should implement the backtracking to extract alignments.
  4. For debugging purpose, try to use a smaller subset (e.g. 100 or 200), it should still work and give some reasonable scores.
  5. In the M step of HMM, please make use of state2count to update the transition prob, and don't forget to clean up the accumulation.
  6. For debugging purpose, try to reduce the GMM component size to 1, it will make the model a bit stabler.
  7. Check alignments and transition probabilities. If the alignment is not balanced (e.g: 1,2,3,3,3,3,3), then something is wrong.
  8. A well-trained transition should have self-recursive probability 0.95, and move to next probability 0.05
- To make your computing faster,
  1. Use vector instead of matrix for std/var in gauss and gmm.py (because only diagonal entries matter)
  2. Use batched matrix computation as much as possible. try not using 'for' statement around vector or scalar in numpy array.
- Other tips:
  1. Don't use np.exp in the E step of GMM. It'll lead to "divide by zero" error later on. Keep it in the log domain only. You may pass GMM tests on gradescope, but not in HMM tests.
  2. In the Viterbi search, you need to backtrack from the last frame in the last state.
  3. You don't need to add extra const (0.5 in self.const) in Gauss because we are using standard deviation, not variance