Instructions: This assignment is intended to introduce you to Hidden Markov Models (HMMs) where you will explore more dynamic programming algorithms such as the forward-backward algorithm and viterbi. You will also gain experience with some of the numerical details that need to be addressed when putting together such a system. You can verify the quality of the MFCC features used in the previous assignments further, and explore the performance difference between DTW-based and HMM-based single-digit speech recognition. Please ask for help if you feel stuck!
Please submit answers to all underlined questions below in the writeup as a PDF file via the course Canvas site. The tex file used to create this PDF has also been provided so that you can enter your answers directly into this document. Additionally, please run through and execute all of the cells of the ipython notebook (including the parts you added which are marked by `# TODO`) and submit the notebook via Canvas.
Lateness policy: Late submissions submitted up to Monday 5/11/20 7:00 pm (the "late deadline") will receive a 20% reduction in credit. Submissions that are later than this will receive some non-zero credit, but we cannot guarantee how much. We cannot guarantee that we will carefully grade or give feedback on submissions after the "late deadline". In addition, you have four free "late days" that you can use throughout the term to extend homework (not project) deadlines without penalty. If you are using any of your "late days" for this assignment, mention that in the comment while submitting this assignment. You should state how many "late days" you are using and how many remaining "late days" you have. The number of late days used for any given homework must be an integer.
Collaboration: You are encouraged to discuss assignments and any aspect of the course material with others, but any material you submit (writeup, code, figures) should be produced on your own.

*** Before starting this homework, you'll need to make sure you have python3 and the following packages on your system: `numpy, scipy, jupyter, matplotlib, pysoundfile`. You may use any installer of your choice. Alternatively, you an follow the steps from HW2 to create a python3 environment and install the required packages using Miniconda.

Once everything is set up, you can open a notebook with the command: `jupyter notebook hw3.ipynb`. Make sure that the kernel is set to Python 3 in the notebook. Please let us know if you need help!

0. Please fill out this questionnaire. This is purely to help us calibrate assignment load and let us know what may need to be made clearer in class. Your responses will receive a small amount of credit, independent of the actual answers.

   (i) Did you collaborate on this assignment, and if so, with whom?
   Yes, With hanqi Zhang.

   (ii) Approximately how many hours did this assignment take to complete?
   10 hours

   (iii) On a scale of 1 to 5 (where 1 = trivial and 5 = impossible), what was the difficulty of this assignment?
   4

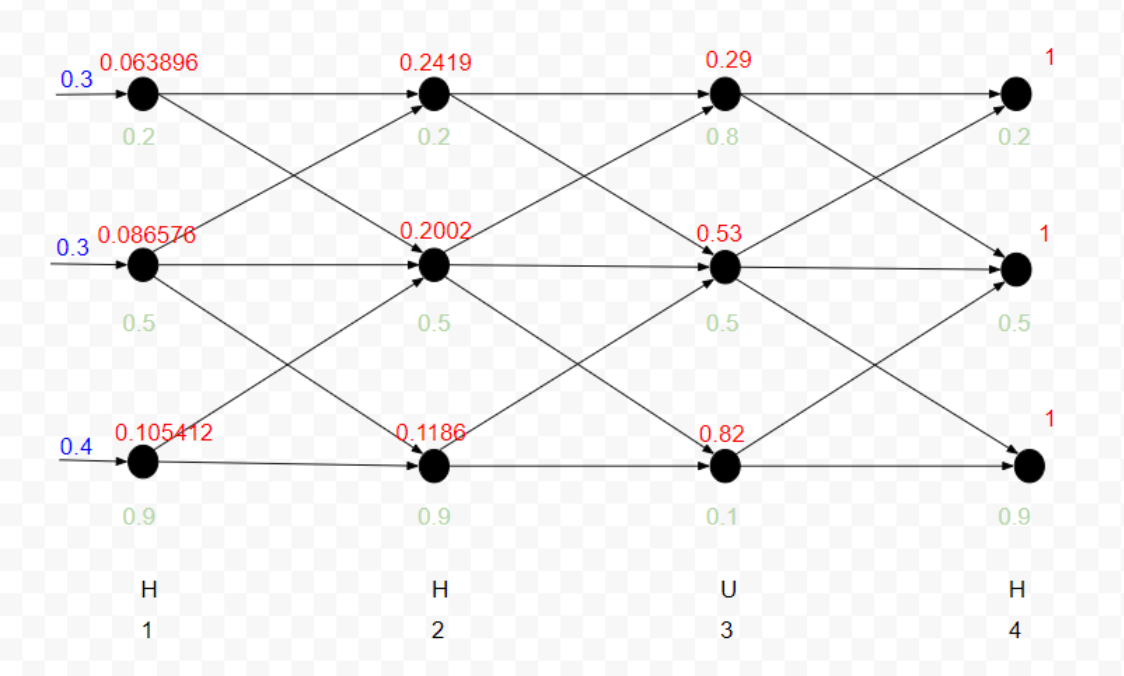
   (iv) On a scale of 1 to 5 (where 1 = useless and 5 = essential), how useful has this assignment been to your understanding of the material?
   4

1. Hidden Markov Models (HMMs)

   (a) Show that if any elements of the parameters π or $\mathbf{A}$ for an HMM are initially set to zero, then those elements will remain zero in all subsequent updates of the EM algorithm.
   1. If $\pi_s = 0$, then $\hat{\pi_s} = \gamma_1(s) = \sum_{j=1}^{N} \xi_1(i,j) = \sum_{j=1}^{N} \alpha_1(s) a_{s,j} b_j(o_2) \beta_2(j) / p(O\|\lambda) = 0$
   since $\alpha_1(s) = \pi_s * b_s(o1) = 0$, therefore, π will remain the same.
   2. If $A_{i,j} = 0$, then $\hat{A_{i,j}} = \sum_{t=1}^{T} \xi_t(i,j) / \sum_{t=1}^{T-1} \gamma_t(i,j) = 0$
   since $\xi_t(i,j) = \alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j) / p(O\|\lambda) = 0$, because $a_{i,j} = 0$, therefore, $A_{i,j}$ will remain the same as 0.

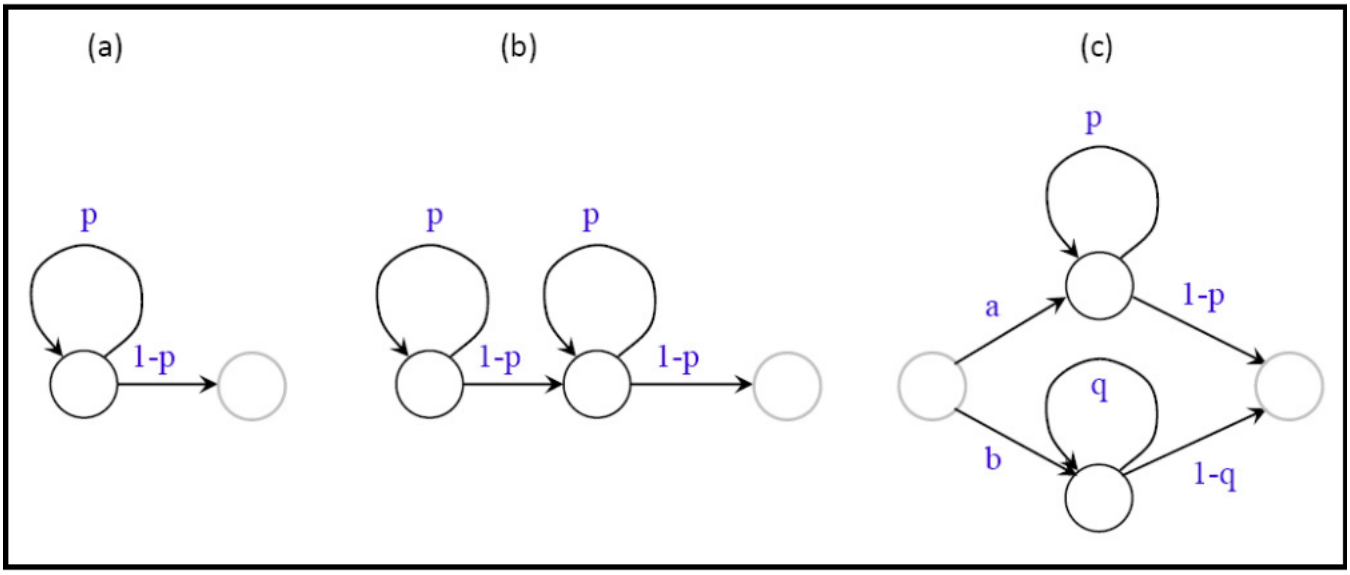
(b) For the weather-mood HMM example shown in class, we manually ran the forward algorithm on a trellis for the observation sequence $O = HHUH$. Run the backward algorithm in the same way and submit your marked-up trellis, with the value of β indicated at each point, and final value for $p(O|\lambda)$ (which should be the same as that found in class with the forward algorithm).

$\beta_3(1)$ = 0.7*0.2*1+0.3*0.5*1 = 0.29
$\beta_3(2)$ = 0.3*0.2*1+0.4*0.5*1+0.3*0.9*1 = 0.53
$\beta_3(3)$ = 0.8*0.9*1+0.2*0.5*1 = 0.82
$\beta_2(1)$ = 0.7*0.8*0.29+0.3*0.53*0.5 = 0.2419
$\beta_2(2)$ = 0.3*0.29*0.8 + 0.5*0.53*0.4+0.3*0.1*0.82 = 0.2002
$\beta_2(3)$ = 0.8*0.1*0.82+0.2*0.5*0.53 = 0.1186
$\beta_1(1)$ = 0.7*0.2419*0.2+0.3*0.5*0.2002=0.0638
$\beta_1(2)$ = 0.2419*0.2*0.3+0.2002*0.5*0.4+0.3*0.1186*0.9 = 0.0866
$\beta_1(3)$ = 0.2*0.5*0.2002 + 0.8*0.9*0.1186 = 0.1054
P(O|λ) = 0.3*0.2*0.0638+0.3*0.5*0.0866+0.4*0.1054*0.9 = 0.055



(c) As discussed in lecture, one of the limitations of HMMs is their geometric state duration distributions. One way of getting around this problem is to expand the state transition diagram in certain ways. In particular, for a given state whose duration distribution we wish to modify, we can expand it into multiple "tied" states, i.e. states with identical output distributions, with specific transition structures depending on the desired duration distribution. To see this, consider the 3 HMM transition diagrams shown below. The states with black outlines all have the same observation (emission) distributions, while the initial/final states in gray are non-emitting. HMM (a) has a single state, with probability p of staying in that state and probability 1-p of exiting the state (and therefore exiting the HMM). HMM (b) has two such states in series, and HMM (c) has two parallel paths with different transition probabilities (but still the same emission distributions).

Give an expression for the probability mass function of duration for each model. That is, if X is the number of time steps from the start state to the exit state, what is the PMF of X, $p(X)$, for each case?
Optional: Generalize the result to any number N of states connected in parallel or in series, and/or plot some distributions of such models in python or another tool.



a/
P(X=x) $= p^{x-1}(1-p)$
b/
P(X = x) $= (x-1)p^{x-2}(1-p)^2$
3/
P(X=x) $= ap^{x-1}(1-p) + bq^{x-1}(1-q)$
Generalization:
b/ P(X = x) $= \binom{X-N+N-1}{N-1}p^{x-N}(1-p)^N = \binom{X-1}{N-1}p^{x-N}(1-p)^N$
c/ parallel:
P(X=x) $= \sum_i a_i p^{x-2}(1-p)$

2. Single Digit Recognition with HMMs

For this part you will use a basic HMM-based recognizer to do single-digit recognition. You will use a data set consisting of training and test utterances containing isolated digits (55 each of 0-9, where '0" is always pronounced "oh" and not "zero"). The data and scripts are to be downloaded from the canvas site.

(a) Complete the implementations, in the ipynb, for the following (Note: all `# TODO`'s are in the code and can be completed in a single line):

- forward, backward, and Viterbi algorithms by filling in the 'forward', 'backward', and 'viterbi' functions
- log probability of the observed sequence (denoted `log_prob`)
- state posteriors (denoted `gamma`)
- state transition probabilities (i to j at time t) (denoted `xi`)
- means and (diagonal) covariances (denoted `self.mu` and `self.sigma`)

For detail, see the code in .ipynb

(b) Experiment with HMM single-digit recognition and see if you can improve the error rate. At least, experiment with different values for the number of states. Optionally, try changing other aspects: parameter initialization, convergence criteria, HMM topology, or any other aspect that you find interesting to experiment with. Run the corresponding added/modified cells in the notebook before submitting.

Forward accuracy of 15 states, 15 iterations: 0.9786 viterbi accuracy of 15 states, 15iterations: 0.9786

Forward accuracy of 10 states, 10 iterations: 0.9643 viterbi accuracy of 10 states, 10iterations: 0.9643

Forward accuracy of 5 states, 10 iterations: 0.8268 viterbi accuracy of 5 states, 10 iterations: 0.8268

(c) Describe in 1-2 paragraphs what you did, the results you obtained, how they compared to your results with DTW, and any other comments about your experiments (e.g. the trade-off between performance and efficiency, number of iterations required, types of errors, ...). There will be some extra credit for the best final

performance. (Note that, like in HW2, we are again "cheating" by tuning on the test set...)

To summarize this HMM-based model.

I write the forward and backward algorithms to compute the $\alpha and \beta$, which is used to calculate the posterior distribution of transition probability $\xi(i,j)$ and also the total probability: $P(O|\lambda) = \sum P(q_t = i, q_{t+1} = j|O,\lambda)$
Note that : $\xi(i,j) = \mathrm{P}(q_t = i, q_{t+1} = j, O|\lambda) = \mathrm{P}(q_t = i, q_{t+1} = j, O|\lambda)/\ \mathrm{P}(q_t = i, q_{t+1} = j|O,\lambda)$
This is the forward, backward, and the score function in python file.

Then the M step is to update the μ and σ using the Baum-Welch equation to update the parameters using MLE method in the mixture model.

Note that:

$\gamma_{il}(t) =$ (lth component of ith mixture that generate oberservation O) $= P(q = i, x = l|O,\lambda) = P(O, q = i|\lambda)/P(O|\lambda) = \alpha * \beta/P(O|\lambda)$ and we use γ to update the parameters in mixture model.

Then repeat E and M step with multiple iterations until convergence.The result is very good after 10 iterations with 10 hidden states.

Comparison with DTW algorithm:

Markov modeling does not need explicit time alignment.It only includes a transition matrix, prior and observation probability.And this algorithm predict the result with the maximum observation probability. DTW will predict the observation by using the warping function to test the distance between test sequence and the references. And then pick the most similar one as the recognition.

DTW has a simpler training process and need less training examples compared to HMM, but it need more time during testing. HMM will need more training time but less time during recognize. The HMM performs better in accuracy.