Name: Marco Ravelo

Problem Set 3: HMMs
The University of Texas at Austin
CS395T: Spoken Language Technologies / Fall 2022
Instructor: David Harwath

---

*some work for problem 1 was done on paper — scanned pages are at the end of this file
*code is presented in code blocks in the appropriate section

1.1. List all of the possible hidden state sequences that could have produced the observation sequence *HTT*.

1: $(s_1 = H,\ s_2 = T,\ s_3 = T,\ s_4 = end)$
2: $(s_1 = H,\ s_3 = T,\ s_3 = T,\ s_4 = end)$
3: $(s_1 = H,\ s_1 = T,\ s_3 = T,\ s_4 = end)$

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

$(s_1 = H,\ s_3 = T,\ s_3 = T,\ s_4 = end)$

1.3. What is $P(HTT|\lambda)$?

$$P(HTT|\lambda) = 0.08203125 \approx 8.203\,\%$$

Here is the code that I wrote to compute the forward algorithm for exercise 1:

```
# define vars
import numpy as np

S = np.array([1, 2, 3, 4]) # state set
X = np.array(['H', 'T', 'T', 'end']) # observation set
A = np.array([[0.25, 0.25, 0.50, 0.00],
              [0.00, 0.50, 0.50, 0.00],
              [0.00, 0.00, 0.75, 0.25],
              [0.00, 0.00, 0.00, 1.00]]) # state-transition set (transition from state x to state
y) -> alpha[state x, state y]

B = np.array([[0.50, 0.75, 0.25, 1.00],
              [0.50, 0.25, 0.75, 1.00]]) # state-observed set -> B[‘H’ || ‘T’ || ‘end’, state]

P = np.array([1.0, 0.0, 0.0, 0.0]) # init-state set

alpha = np.array([[0.00, 0.00, 0.00, 0.00],
                  [0.00, 0.00, 0.00, 0.00],
                  [0.00, 0.00, 0.00, 0.00],
                  [0.00, 0.00, 0.00, 0.00]]) # -> A[time, state]
```

```
def get_b(x, state): # get prob of observation x at state i
    prob = 0.0
    if (x == 'H'):
        prob = B[0, state]
    elif (x == 'T'):
        prob = B[1, state]
    elif (x == 'end'):
        prob = 1.0
    return prob
```

```
# forward algorithm

# init:
for i in range(0, len(S)):
    val = get_b(X[0], i) * P[i]
    alpha[0, i] = val

# induction:
# for each time step from 0 -> 3
for t in range(0, len(S) - 1):
    # for each state i from 0 -> 3
    for i in range(len(S)):
        sum_array = []
        # for each state j from 0 -> 3
        for j in range(len(S)):
            sum_array.append(alpha[t, j] * A[j, i])
        sum_total = np.sum(sum_array)
        alpha[t + 1, i] = sum_total * get_b(X[t + 1], i)

print ("alpha matrix = ", alpha)

T = 3
for i in range(len(S)):
    sum_array = []
    sum_array.append(alpha[T, i])

# termination:
final = np.sum(sum_array)
print ("final: ", final)

# in this case, t = 2 is actually t = 1 since we start from t = 0 and not t = 1.
# state 1 is still s = 1
alpha_s1_t1 = alpha[1, 1]
print ("alpha_s1_t1: ", alpha_s1_t1)
```

code output:

```
alpha matrix = [[0.5        0.         0.         0.        ]
                [0.0625     0.03125    0.1875     0.        ]
                [0.0078125  0.0078125  0.140625   0.046875  ]
                [0.00195312 0.00585938 0.11328125 0.08203125]]
final:  0.08203125
alpha_s1_t1:  0.03125
```

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

$$alpha\ @\ state\ =\ 1\ and\ t\ =\ 2:\ 0.03125\ \approx\ 3.125\ \%$$

2.1. Computing class likelihoods:

code:

```
# load in audio files
# convert to mel-logspec tensor
# compute phone likelihoods for each

burt_phone_likelihoods = compute_phone_likelihoods(model,
load_audio_to_melspec_tensor(wavpath="burt.wav"))
fee_phone_likelihoods = compute_phone_likelihoods(model,
load_audio_to_melspec_tensor(wavpath="fee.wav"))
pea_phone_likelihoods = compute_phone_likelihoods(model,
load_audio_to_melspec_tensor(wavpath="pea.wav"))
rock_phone_likelihoods = compute_phone_likelihoods(model,
load_audio_to_melspec_tensor(wavpath="rock.wav"))
see_phone_likelihoods = compute_phone_likelihoods(model,
load_audio_to_melspec_tensor(wavpath="see.wav"))
she_phone_likelihoods = compute_phone_likelihoods(model,
load_audio_to_melspec_tensor(wavpath="she.wav"))

print ("burt_phone_likelihoods.shape: ", burt_phone_likelihoods.shape)
print ("fee_phone_likelihoods.shape: ", fee_phone_likelihoods.shape)
print ("pea_phone_likelihoods.shape: ", pea_phone_likelihoods.shape)
print ("rock_phone_likelihoods.shape: ", rock_phone_likelihoods.shape)
print ("see_phone_likelihoods.shape: ", see_phone_likelihoods.shape)
print ("she_phone_likelihoods.shape: ", she_phone_likelihoods.shape)
```

output:

```
burt_phone_likelihoods.shape:  (100, 48)
fee_phone_likelihoods.shape:  (103, 48)
pea_phone_likelihoods.shape:  (113, 48)
rock_phone_likelihoods.shape:  (81, 48)
see_phone_likelihoods.shape:  (96, 48)
she_phone_likelihoods.shape:  (107, 48)
```

2.2. Implementing the Forward Algorithm:

code:

```
def forward(self, state_likelihoods): # shape is assumed to be (T_timesteps, 48)
        # create B array (T_total_timesteps, N_total_states)
        T_total_timesteps = state_likelihoods.shape[0]
        self.B = np.zeros((T_total_timesteps, self.N_total_states))
        i = 0
        for state in self.labels:
            self.B[:, i] = state_likelihoods[:, state]
            i += 1

        # create alpha matrix (T_total_timesteps, N_total_states)
        alpha_matrix = np.zeros((T_total_timesteps, self.N_total_states))

        # initialization
        t = 0 # time step
        i = 0 # state
        for i in range (self.N_total_states):
            alpha_matrix[t, i] = np.add(self.B[t, i], self.pi[i])

        # induction step
        for t in range(0, T_total_timesteps - 1):
            for i in range (self.N_total_states):
                sum_array = []
                # get sum
                for j in range(self.N_total_states):
                    sum_array.append(np.add(alpha_matrix[t, j], self.A[j, i]))
                # multiply values and set alpha matrix
                value = np.add(logsumexp(sum_array), self.B[t + 1, i])
                alpha_matrix[t + 1, i] = value

        # termination
        return alpha_matrix[T_total_timesteps - 1, self.N_total_states - 1]
```

2.3. Implementing the Viterbi Decoding Algorithm:

code:

```
def viterbi(self, state_likelihoods): # shape is assumed to be (T_timesteps, 48)
        # create B array
        T_total_timesteps = state_likelihoods.shape[0]
        self.B = np.zeros((T_total_timesteps, self.N_total_states))
        i = 0
        for state in self.labels:
            self.B[:,i] = state_likelihoods[:,state]
            i += 1

        # create psi matrix (backtrace)
        psi_matrix = np.zeros((T_total_timesteps, self.N_total_states), dtype=np.int8)
        for i in range(self.N_total_states):
            psi_matrix[0, i] = 0

        # create delta array (T_total_timesteps, N_total_states)
        delta_matrix = np.zeros((T_total_timesteps, self.N_total_states))

        # initialization
        t = 0 # time step
        i = 0 # state
        for i in range (self.N_total_states):
            delta_matrix[t, i] = np.add(self.B[t, i], self.pi[i])

        # induction step
        for t in range(1, T_total_timesteps):
            for i in range (self.N_total_states):
                find_max_array = []
                # get induction values to find max
                for j in range(self.N_total_states):
                    find_max_array.append(np.add(delta_matrix[t - 1, j], self.A[j, i]))
                # find max, multiply values, and set delta matrix
                max = np.max(find_max_array)
                value = np.add(max, self.B[t, i])
                delta_matrix[t, i] = value
                # set psi backtrace value
                psi_matrix[t, i] = int(np.argmax(find_max_array))

        # termination
        output_path = []
        t = T_total_timesteps - 1
        q = int(np.argmax(delta_matrix[t,:]))
        output_path.append(q)
        t -= 1
        while t >= 0:
            q = int(psi_matrix[t + 1, int(q)])
            output_path.append(q)
            t -= 1

        output_path.reverse()
        return output_path
```

2.4. Implementing Viterbi Training:

code:

```
def viterbi_transition_update(self, state_likelihoods): # shape is assumed to be (T_timesteps,48)
        # compute viterbi algorithm
        viterbi_states = self.viterbi(state_likelihoods)

        # create temp A matrix using viterbi output
        temp_A_matrix = np.zeros((self.N_total_states, self.N_total_states))
        # number of times model made transition out of state i
        num_out_states = np.zeros(self.N_total_states)
        t = 0
        for t in range(len(viterbi_states) - 1):
            state_i = viterbi_states[t]
            state_j = viterbi_states[t + 1]
            # number of times transition was made from state i to state j
            temp_A_matrix[state_i, state_j] += 1
            # number of times model made transition out of state i
            num_out_states[state_i] += 1

        # compute log probabilities for A matrix
        for i in range(self.N_total_states):
            for j in range (self.N_total_states):
                numerator = temp_A_matrix[i, j]
                denominator = num_out_states[i]
                self.A[i, j] = np.log((numerator / denominator) + self.eps)

        #print ("A log matrix: ", self.A)
        #print ("A matrix: ",np.exp(self.A))
        #print ("viterbi states: ", viterbi_states)
        return self.A
```

2.5. 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?

	burt HMM	fee HMM	pea HMM	rock HMM	see HMM	she HMM
burt.wav	219.955	-60.167	-17.220	114.755	-68.847	-93.578
fee.wav	-85.011	215.350	182.866	-92.207	191.420	192.044
pea.wav	76.815	254.804	274.087	14.371	241.543	239.840
rock.wav	65.112	-62.829	-7.896	157.203	-61.326	-63.344
see.wav	-152.635	78.483	81.993	-171.339	235.932	134.254
she.wav	-188.118	79.446	89.808	-208.654	126.318	286.063

The orange colored cells show the maximum score for each word using each trainded HMM. As you can see, four out of the six words were correctly identifdied except for “fee” and “see”, which were both misidentified with “pea”. Still, even with this misidentification, “fee” and “see” were the second highest score out of the six words in their HMM column (highlighted in yellow). This was probably due to the phonetic similarity between the words “fee” / “see” and “pea”. You can observe this phonetic similarity by also comparing the scores between “she” and “pea” in the she HMM column, in which “she” was almost misidentified as well.

2.6. 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.

code:

```
# Optimal state sequence:

rock_state_sequence = rock_HMM.viterbi(rock_phone_likelihoods)
print ("rock hidden state sequence:\n", rock_state_sequence)
```

output:

```
rock hidden state sequence:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4,
4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
```

2.7. 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?

code:

```
# Viterbi Update
# new rock HMM
rock_HMM = MyHMM(phones2indices(['sil', 'r', 'aa', 'cl', 'k', 'sil']),
                 np.array([0.5, 0.5, 0.0, 0.0, 0.0, 0.0]),
                 np.array([[0.9, 0.1, 0.0, 0.0, 0.0, 0.0],
                           [0.0, 0.9, 0.1, 0.0, 0.0, 0.0],
                           [0.0, 0.0, 0.9, 0.1, 0.0, 0.0],
                           [0.0, 0.0, 0.0, 0.9, 0.1, 0.0],
                           [0.0, 0.0, 0.0, 0.0, 0.9, 0.1],
                           [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]))
print ("rock transition matrix before update:\n", np.exp(rock_HMM.A))

# rock score before update
rock_score = rock_HMM.forward(rock_phone_likelihoods)
print ("rock score before update: ", rock_score)

print ("-------- UPDATE --------")
```

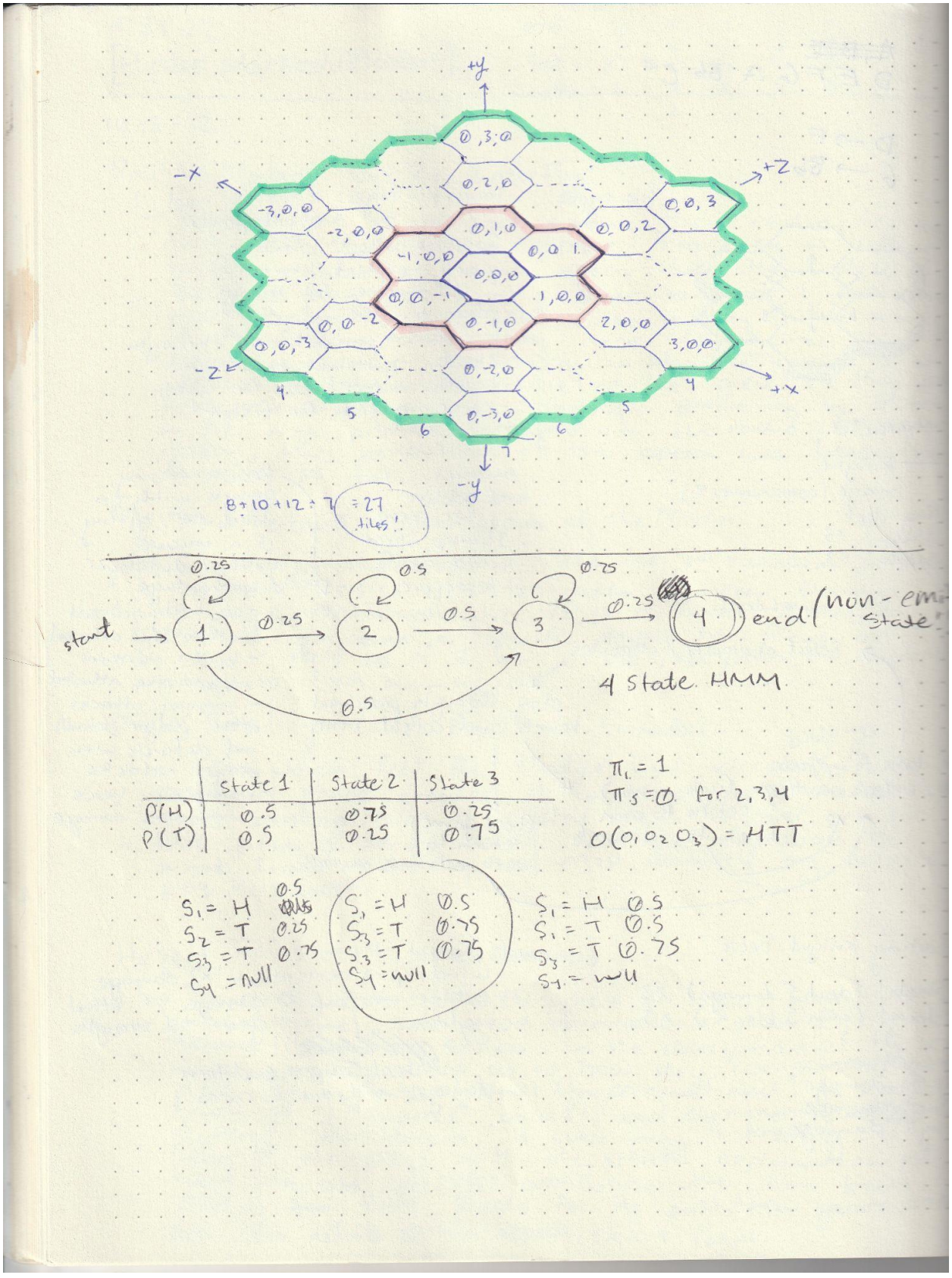
```
# viterbi update
rock_A_matrix = rock_HMM.viterbi_transition_update(rock_phone_likelihoods)
print ("rock transition matrix after update:\n", np.exp(rock_A_matrix))

# rock likelihood after update
rock_score = rock_HMM.forward(rock_phone_likelihoods)
print ("rock score after update: ", rock_score)
```

output:

```
rock transition matrix before update:
 [[0.9 0.1 0.  0.  0.  0. ]
  [0.  0.9 0.1 0.  0.  0. ]
  [0.  0.  0.9 0.1 0.  0. ]
  [0.  0.  0.  0.9 0.1 0. ]
  [0.  0.  0.  0.  0.9 0.1]
  [0.  0.  0.  0.  0.  1. ]]
rock score before update:  156.74317859888868
-------- UPDATE --------
rock transition matrix after update:
 [[0.93333333 0.06666667 0.         0.         0.         0.        ]
  [0.         0.91666667 0.08333333 0.         0.         0.        ]
  [0.         0.         0.92857143 0.07142857 0.         0.        ]
  [0.         0.         0.         0.90909091 0.09090909 0.        ]
  [0.         0.         0.         0.         0.75       0.25      ]
  [0.         0.         0.         0.         0.         1.        ]]
rock score after update:  157.20290893345896
```

The score calculated by the forward algorithm before the viterbi update was 156.743. The score after the update was 157.203. As you can see, the score did go up by about 0.46. The transition matrix, however, had a much more dramatic and noticeable change, the most obvious being the second to last row having its values modified from 0.9 and 0.1 to 0.75 and 0.25 respectively.

*scanned notebook pages



State set $S = \{S_1, S_2, S_3, S_4\}$

observation set $X = \{H, T, T, end\}$

state-transition set $A =$

	1	2	3	4
1	0.25	0.25	0.5	0
2	0	0.5	0.5	0
3	0	0	0.75	0.25
4	0	0	0	1.0

transition from $i \rightarrow j$

state-observation set $B =$

	1	2	3	4
H	0.5	0.75	0.25	0
T	0.5	0.25	0.75	0

init state distribution $\pi = \{1\ 0\ 0\ 0\}$

Q3. what is $P(HTT \mid \lambda)$?

init: $a_1(i) = b_i(o_1) * \pi_i$

$t = 1$

S_1 0.5 * 1.0 = .5

S_2 0.5 * 0.0 = 0

S_3 0.5 * 0.0 = 0

S_4 0.5 * 0.0 = 0

induction: $t = 2$

S_1 0.25 *

S_2 0.25 * $\rightarrow \left(\sum_{j=1}^{4} \alpha_t(j)\, a_{ji}\right) \rightarrow ((0.5 * 0.25) + (0 * 0.5) + (0 * 0) + (0 * 0)) = 0.125$

S_3 0.25 *

S_4 0.25 *

"At this point I realized that it would be much easier and much less error-prone to just write a script to compute this for me."

See below ↓

*Ignore the "see below" I wrote, the code that computes this is in exercise 1 of this document.