Student Name: Pengcheng Xu

Collaboration Statement:

Total hours spent: 5 hrs

I discussed ideas with these individuals:

- I finished it on my own
- . . .

I consulted the following resources:

- Course slides
- Course website
- . . .

By submitting this assignment, I affirm this is my own original work that abides by the course collaboration policy.

Links: [CP5 instructions] [collab. policy]

Contents

Forward Algorithm Implementation

```
def run_forward_algorithm(log_pi_K, log_A_KK, log_lik_TK):


  # Destruct T and K
  T, K = log_lik_TK.shape

  # Allocate results
  log_pdf_x_1toT = 0.0
  alpha_TK = np.zeros((T,K))

  # TODO base case update at t=0, Using the formula provided
  log_pdf_x_1toT += logsumexp(log_pi_K + log_lik_TK[0,:])    # ok
  alpha_TK[0, :] = np.exp(log_pi_K + log_lik_TK[0, :] - log_pdf_x_1toT) # ok


  for t in range(1, T):
    # TODO recursive update for t=1, ... T-1

     # compute denominator
    log_denom = logsumexp(log_A_KK + log_lik_TK[t, :] + np.log(alpha_TK[t-1,:])[:, np.newaxis])

    # update log_pdf_x_1toT
    log_pdf_x_1toT += log_denom

    # Using log property, and put numerator and denominator together
    alpha_TK[t, :] = np.exp( log_lik_TK[t, :] + logsumexp( np.log(alpha_TK[t-1, :])[:, np.newaxis
                                                ] + log_A_KK, axis = 0) - log_denom )

  return alpha_TK, log_pdf_x_1toT
```

Viterbi Algorithm Implementation

```
def run_viterbi_algorithm(log_pi_K, log_A_KK, log_lik_TK):
# destruct T and K
  T, K = log_lik_TK.shape

  # Allocate array for storing log joint probabilities
  log_w_TK = np.zeros((T,K))
  # Allocate array for storing "backpointers"
  b_TK = -1 * np.ones((T, K), dtype=np.int32)

  # TODO base case update of log_w_TK and b_TK at t=0
  log_w_TK[0, :] = log_lik_TK[0, :] + log_pi_K    # okay
  b_TK[0, :] = -1
  for t in range(1, T):
    # TODO base case update of log_w_TK and b_TK from t = 1 to  T- 1, Using formulas given in the
                                                    slide
    log_w_TK[t, :] = log_lik_TK[t, :] +  np.max(log_A_KK + log_w_TK[t-1, :][:, np.newaxis], axis=
                                                  0 )
    b_TK[t, :] = np.argmax(log_A_KK + log_w_TK[t-1, :][:, np.newaxis], axis = 0)

  # Allocate array for storing estimated z sequence
  zhat_T = np.zeros(T, dtype=np.int32)
  # TODO Update at final timestep
  zhat_T[-1] = np.argmax(log_w_TK[T-1, :])

  for t in range(T-2, -1, -1): # count from T-2, .... 1, 0 inclusive
    # TODO Update at t given zhat at t+1, backtrace back pointers
    zhat_T[t] = b_TK[t+1, zhat_T[t+1]]

  # TODO compute joint probability of entire sequence
# log_pdf_x_and_zhat should be the max value in the vector log_w_TK[T-1]
  hmm_log_pdf_x_and_zhat = np.max(log_w_TK[T-1, :])

  return zhat_T, hmm_log_pdf_x_and_zhat
```