Human-Oriented Robotics
Prof. Dr. Kai Arras, Social Robotics Lab
Lab instructors: Timm Linder, Luigi Palmieri, Billy Okal

Winter term 2014/2015
University of Freiburg
Department of Computer Science

Submission: Send your solution via email to `palmieri@informatik.uni-freiburg.de` until January 20, 2015 with subject "[`exercises`] `Sheet 8`". All files (Matlab scripts, exported figures, hand-written notes in pdf/jpg format) should be compressed into a single zip file named `lastname_sheet8.zip`.

Exercise 8: Hidden Markov Models

For this exercise, you will need to download the Matlab frame `RunHMM.m` from the course website. Note for Octave users: we will use the built-in Matlab command `hmmgenerate`.

Exercise 8.1: Forward-Backward Algorithm

In this exercise you will implement the forward-backward algorithm, and compare the inference results for filtering and smoothing. The algorithm has a compact matrix implementation which makes it particularly simple to implement in Matlab/Octave.

Let us remember our HMM notation: hidden variables $\mathbf{x}$ have $s \in \{1, ..., S\}$ states, the transition and observation models are described by matrices A and E of dimension $S \times S$ and $S \times O$, respectively, where O is the number of observation symbols (here we will assume $S = O = 3$).

The algorithm uses the probability $p(\mathbf{z}_k \mid \mathbf{x}_k = s)$ which specifies how likely it is that state s causes $\mathbf{z}_k$ to appear. Following our definition of matrix E this corresponds to a column of E. For mathematical convenience we place these values into a $S \times S$ diagonal matrix D. If, for instance, $p(\mathbf{z}_k \mid \mathbf{x}_k = 1) = (0.9 \;\; 0.2)^T$ (from the umbrella example), then

$$D_k = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$$

(**Hint:** to do this use `diag`). Now, using column vectors to represent the forward and backward probabilities α_k and β_k, all the computations become simple matrix-vector operations. The forward step becomes

$$\alpha_{k+1} = \eta \cdot D_k \cdot A^T \cdot \alpha_k \tag{1}$$

and the backward step becomes

$$\beta_k = A \cdot D_k \cdot \beta_{k+1} \tag{2}$$

The initial conditions are $\alpha_0 = p(\mathbf{x}_0)$, where $p(\mathbf{x}_0)$ is a $S \times 1$ prior distribution, and $\beta_K = \mathbf{1}$, a $S \times 1$ vector of 1's, where K is the length of the sequence. HMM parameters and additional explanations are given in the frame `RunHMM.m`. Proceed as follows:

a) Implement the forward algorithm and plot the probabilities for each state s over time. Normalize the α's for each step in order them to be proper probability distributions.

b) Implement the forward-backward algorithm and plot the probabilities for each state s over time.

c) Compare the results visually, try different random seeds and rerun the algorithms. Familiarize yourself with the algorithms' behavior.

Use `nstates` $\times$ `nobservations+1` matrices for α, β and the smoothed probabilities (the `+1` is for the prior α_0). Iterate in both algorithms over the index range `2` to `nobservations+1` corresponding to time steps $k = 1, ..., K$.

Exercise 8.2: Viterbi Algorithm

The Viterbi algorithm computes the most likely state sequence $\mathbf{x}^*$ given a sequence of observations $\{\mathbf{z}_1, ..., \mathbf{z}_K\}$. For each step $k = 1, ..., K$ the algorithm recursively calculates the $S \times 1$ probability distribution μ_k. We will consider the matrix form of the algorithm which is obtained by "diagonalizing" μ_k into the $S \times S$ diagonal matrix M_k (again using `diag`) and using the following update equation

$$\mu_{k+1} = D_k \cdot \max\left(A^T \cdot M_k\right) \tag{3}$$

The initial distribution μ_1 is obtained as $\mu_1 = p(z_1 \mid x_1) \cdot p(x_0)$ which in matrix form is

$$\mu_1 = D_1 \cdot p(x_0). \tag{4}$$

$p(x_0)$ is again the $S \times 1$ prior distribution.

Use `[m,i] = max(B,[],2)` to vectorize the maximum computation and obtain both the maxima of the rows of `B` and the indices of those maxima (you might consult `help max` for more information). You need to store these indices to be able to reconstruct the most likely state sequence.

After the computation of $\mu_{1..K}$ and the indices of the respective maxima, write an algorithm that iteratively reconstructs the most likely state sequence $\mathbf{x}^*$. First, find the maximum of μ_K, s_K^*, which is already the last state of $\mathbf{x}^*$. Then, find the most likely predecessor of s_K^*, s_{K-1}^*, which is the second last state of $\mathbf{x}^*$. And so on, repeat until $k = 1$.

a) Implement the Viterbi algorithm and plot the most likely state sequence. For μ, use an `nstates` $\times$ `nobservations` matrix where each column represents the vector μ_k of a particular step.

b) Try different random seeds and find a case where the Viterbi algorithm fails to reconstruct the ground truth sequence (e.g. misses a relatively long subsequence of a state). Explain the reason and fix the problem by making changes to the matrix E.