IDIAP COMMUNICATION

IDIAP

Martigny - Valais - Suisse

EPFL lab session 2/2 : Introduction to Hidden Markov Models

Sacha Krstulović

IDIAP–Com 01-07

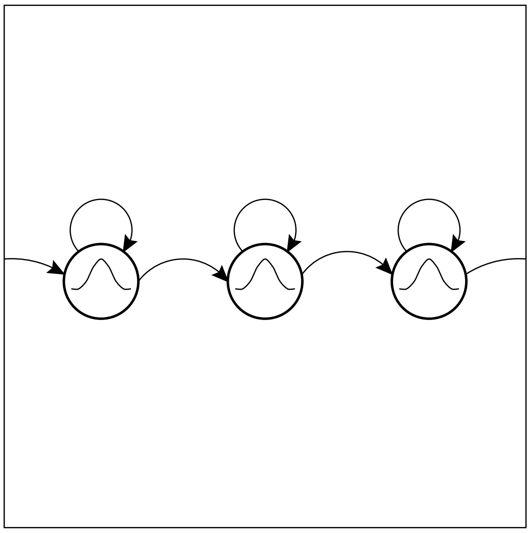
January 2001

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 − 27 − 721 77 11
fax +41 − 27 − 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

Ecole Polytechnique Fédérale de Lausanne

Lab session 2 :
Introduction to Hidden Markov Models



Course : Speech processing and speech recognition

Teacher : Prof. Hervé Bourlard `bourlard@idiap.ch`

Assistants : Sacha Krstulović `sacha@idiap.ch`
Mathew Magimai-Doss `mathew@idiap.ch`

Guidelines

The following lab manual is structured as follows :

- each section corresponds to a theme
- each subsection corresponds to a separate experiment.

The subsections begin with useful formulas and definitions that will be put in practice during the experiments. These are followed by the description of the experiment and by an example of how to realize it in MATLAB.

If you follow the examples literally, you will be able to progress into the lab session without worrying about the experimental implementation details. If you have ideas for better MATLAB implementations, you are welcome to put them in practice provided you don't loose too much time : remember that a lab session is no more than 3 hours long.

The subsections also contain questions that you should think about. Corresponding answers are given right after, in case of problem. You can read them right after the question, *but*: the purpose of this lab is to make you

Think !

If you get lost with some of the questions or some of the explanations, DO ASK the assistants or the teacher for help : they are here to make the course understood. There is no such thing as a stupid question, and the only obstacle to knowledge is laziness.

Have a nice lab;

Teacher & Assistants

Before you begin...

If this lab manual has been handed to you as a hardcopy :

1. get the lab package from
   `ftp.idiap.ch/pub/sacha/labs/Session2.tgz`
2. un-archive the package :
   ```
   % gunzip Session2.tgz
   % tar xvf Session2.tar
   ```
3. change directory :
   ```
   % cd session2
   ```
4. start MATLAB :
   ```
   % matlab
   ```

Then go on with the experiments...

This document was created by : Sacha Krstulović (sacha@idiap.ch).
This document is currently maintained by : Sacha Krstulović (sacha@idiap.ch). Last modification on November 27, 2001.
This document is part of the package Session2.tgz available by ftp as : ftp.idiap.ch/pub/sacha/labs/Session2.tgz .

Contents

1 Preamble 1

2 Generating samples from Hidden Markov Models 4

3 Pattern recognition with HMMs 6
3.1 Likelihood of a sequence given a HMM 6
3.2 Bayesian classification 8
3.3 Maximum Likelihood classification 9

4 Optimal state sequence 10

5 Training of HMMs 13

1 Preamble

Useful formulas and definitions :

- a *Markov chain* or *process* is a sequence of events, usually called *states*, the probability of each of which is dependent only on the event immediately preceding it.

- a *Hidden Markov Model* (HMM) represents stochastic sequences as Markov chains where the states are not directly observed, but are associated with a probability density function (pdf). The generation of a random sequence is then the result of a random walk in the chain (i.e. the browsing of a random sequence of states $Q = \{q_1, \cdots q_K\}$) and of a draw (called an *emission*) at each visit of a state.

  The sequence of states, which is the quantity of interest in speech recognition and in most of the other pattern recognition problems, can be observed only *through* the stochastic processes defined into each state (i.e. you must know the parameters of the pdfs of each state before being able to associate a sequence of states $Q = \{q_1, \cdots q_K\}$ to a sequence of observations $X = \{x_1, \cdots x_K\}$). The true sequence of states is therefore *hidden* by a first layer of stochastic processes.

  HMMs are *dynamic models*, in the sense that they are specifically designed to account for some macroscopic structure of the random sequences. In the previous lab, concerned with *Gaussian Statistics and Statistical Pattern Recognition*, random sequences of observations were considered as the result of a series of *independent* draws in one or several Gaussian densities. To this simple statistical modeling scheme, HMMs add the specification of some *statistical dependence* between the (Gaussian) densities from which the observations are drawn.

- *HMM terminology* :

  – the *emission probabilities* are the pdfs that characterize each state q_i, i.e. $p(x|q_i)$. To simplify the notations, they will be denoted $b_i(x)$. For practical reasons, they are usually Gaussian or combinations of Gaussians, but the states could be parameterized in terms of any other kind of pdf (including discrete probabilities and artificial neural networks).

  – the *transition probabilities* are the probability to go from a state i to a state j, i.e. $P(q_j|q_i)$. They are stored in matrices where each term a_{ij} denotes a probability $P(q_j|q_i)$.

  – *non-emitting initial and final states*: if a random sequence $X = \{x_1, \cdots x_K\}$ has a finite length K, the fact that the sequence begins or ends has to be modeled as two additional discrete events. In HMMs, this corresponds to the addition of two *non-emitting states*, the initial state and the final state. Since their role is just to model the "start" or "end" events, they are not associated with some emission probabilities.

    The transitions starting from the initial state correspond to the modeling of an *initial state distribution* $P(I|q_j)$, which indicates the probability to start the state sequence with the emitting state q_j.

The final state usually has only one non-null transition that loops onto itself with a probability of 1 (it is an *absorbent state*), so that the state sequence gets "trapped" into it when it is reached.

– *ergodic versus left-right HMMs*: a HMM allowing for transitions from any emitting state to any other emitting state is called an *ergodic HMM*. Alternately, an HMM where the transitions only go from one state to itself or to a unique follower is called a *left-right HMM*.

Values used throughout the experiments:

The following 2-dimensional Gaussian densities will be used to model simulated vowel observations, where the considered features are the two first formants:

Density $\mathcal{N}_{/a/}$: $\quad \mu_{/a/} = \begin{bmatrix} 730 \\ 1090 \end{bmatrix} \quad \Sigma_{/a/} = \begin{bmatrix} 1625 & 5300 \\ 5300 & 53300 \end{bmatrix}$

Density $\mathcal{N}_{/e/}$: $\quad \mu_{/e/} = \begin{bmatrix} 530 \\ 1840 \end{bmatrix} \quad \Sigma_{/e/} = \begin{bmatrix} 15025 & 7750 \\ 7750 & 36725 \end{bmatrix}$

Density $\mathcal{N}_{/i/}$: $\quad \mu_{/i/} = \begin{bmatrix} 270 \\ 2290 \end{bmatrix} \quad \Sigma_{/i/} = \begin{bmatrix} 2525 & 1200 \\ 1200 & 36125 \end{bmatrix}$

Density $\mathcal{N}_{/o/}$: $\quad \mu_{/o/} = \begin{bmatrix} 570 \\ 840 \end{bmatrix} \quad \Sigma_{/o/} = \begin{bmatrix} 2000 & 3600 \\ 3600 & 20000 \end{bmatrix}$

Density $\mathcal{N}_{/y/}$: $\quad \mu_{/y/} = \begin{bmatrix} 440 \\ 1020 \end{bmatrix} \quad \Sigma_{/y/} = \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix}$

(Those densities have been used in the previous lab session.) They will be combined into Markov Models that will be used to model some observation sequences. The resulting HMMs are described in table 1.

The parameters of the densities and of the Markov models are stored in the file `data.mat`. A Markov model named, e.g., `hmm1` is stored as an object with fields `hmm1.means`, `hmm1.vars` and `hmm1.trans`, and corresponds to the model HMM1 of table 1. The `means` fields contains a list of mean vectors; the `vars` field contains a list of variance matrices; the `trans` field contains the transition matrix; e.g to access the mean of the 3^{rd} state of `hmm1`, use:

```
>> hmm1.means{3}
```

The initial and final states are characterized by an empty mean and variance value.

Preliminary Matlab commands:

Before realizing the experiments, execute the following commands:

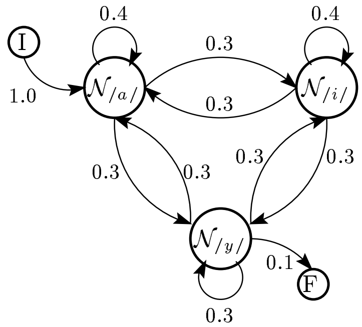
```
>> colordef none; % Set a black background for the figures
>> load data; % Load the experimental data
>> whos % View the loaded variables
```

Emission probabilities	**Transition matrix**	**Sketch of the model**

HMM1:

- state 1: initial state
- state 2: Gaussian $\mathcal{N}_{/a/}$
- state 3: Gaussian $\mathcal{N}_{/i/}$
- state 4: Gaussian $\mathcal{N}_{/y/}$
- state 5: final state

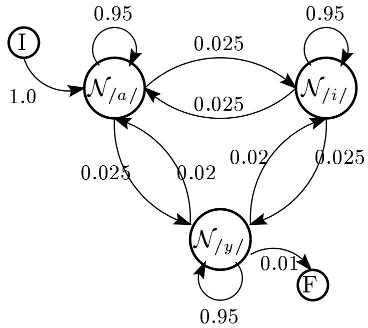
$$\begin{bmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 & 0.0 \\ 0.0 & \mathbf{0.4} & \mathbf{0.3} & \mathbf{0.3} & 0.0 \\ 0.0 & \mathbf{0.3} & \mathbf{0.4} & \mathbf{0.3} & 0.0 \\ 0.0 & \mathbf{0.3} & \mathbf{0.3} & \mathbf{0.3} & \mathbf{0.1} \\ 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{1.0} \end{bmatrix}$$



HMM2:

- state 1: initial state
- state 2: Gaussian $\mathcal{N}_{/a/}$
- state 3: Gaussian $\mathcal{N}_{/i/}$
- state 4: Gaussian $\mathcal{N}_{/y/}$
- state 5: final state

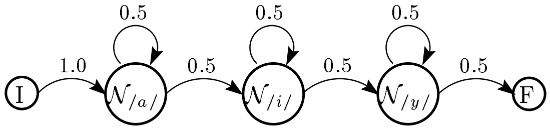
$$\begin{bmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 & 0.0 \\ 0.0 & \mathbf{0.95} & \mathbf{0.025} & \mathbf{0.025} & 0.0 \\ 0.0 & \mathbf{0.025} & \mathbf{0.95} & \mathbf{0.025} & 0.0 \\ 0.0 & \mathbf{0.02} & \mathbf{0.02} & \mathbf{0.95} & \mathbf{0.01} \\ 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{1.0} \end{bmatrix}$$



HMM3:

- state 1: initial state
- state 2: Gaussian $\mathcal{N}_{/a/}$
- state 3: Gaussian $\mathcal{N}_{/i/}$
- state 4: Gaussian $\mathcal{N}_{/y/}$
- state 5: final state

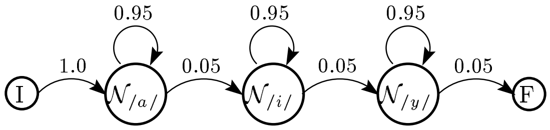
$$\begin{bmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 & 0.0 \\ 0.0 & \mathbf{0.5} & \mathbf{0.5} & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{0.5} & \mathbf{0.5} & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.5} & \mathbf{0.5} \\ 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{1.0} \end{bmatrix}$$



HMM4:

- state 1: initial state
- state 2: Gaussian $\mathcal{N}_{/a/}$
- state 3: Gaussian $\mathcal{N}_{/i/}$
- state 4: Gaussian $\mathcal{N}_{/y/}$
- state 5: final state

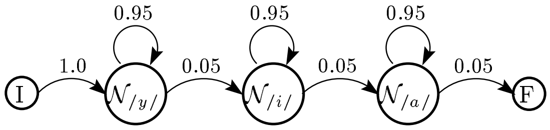
$$\begin{bmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 & 0.0 \\ 0.0 & \mathbf{0.95} & \mathbf{0.05} & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{0.95} & \mathbf{0.05} & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.95} & \mathbf{0.05} \\ 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{1.0} \end{bmatrix}$$



HMM5:

- state 1: initial state
- state 2: Gaussian $\mathcal{N}_{/y/}$
- state 3: Gaussian $\mathcal{N}_{/i/}$
- state 4: Gaussian $\mathcal{N}_{/a/}$
- state 5: final state

$$\begin{bmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 & 0.0 \\ 0.0 & \mathbf{0.95} & \mathbf{0.05} & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{0.95} & \mathbf{0.05} & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.95} & \mathbf{0.05} \\ 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{1.0} \end{bmatrix}$$



HMM6:

- state 1: initial state
- state 2: Gaussian $\mathcal{N}_{/a/}$
- state 3: Gaussian $\mathcal{N}_{/i/}$
- state 4: Gaussian $\mathcal{N}_{/e/}$
- state 5: final state

$$\begin{bmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 & 0.0 \\ 0.0 & \mathbf{0.95} & \mathbf{0.05} & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{0.95} & \mathbf{0.05} & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.95} & \mathbf{0.05} \\ 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{1.0} \end{bmatrix}$$

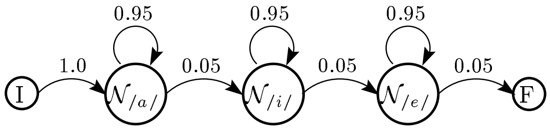


Table 1: List of the Markov models used in the experiments.

2 Generating samples from Hidden Markov Models

Experiment :

Generate a sample X coming from the Hidden Markov Models HMM1, HMM2 and HMM4. Use the function `genhmm` (`>> help genhmm`) to do several draws with each of these models. View the resulting samples and state sequences with the help of the functions `plotseq` and `plotseq2`.

Example :

Do a draw :

```
>> [X,stateSeq] = genhmm(hmm1);
```

Use the functions `plotseq` and `plotseq2` to picture the obtained 2-dimensional data. In the resulting views, the obtained sequences are represented by a yellow line where each point is overlaid with a colored dot. The different colors indicate the state from which any particular point has been drawn.

```
>> figure; plotseq(X,stateSeq); % View of both dimensions as separate sequences
```

This view highlights the notion of sequence of states associated with a sequence of sample points.

```
>> figure; plotseq2(X,stateSeq,hmm1); % 2D view of the resulting sequence,
                                      % with the location of the Gaussian states
```

This view highlights the spatial repartition of the sample points.

Draw several new samples with the same parameters and visualize them :

```
>> clf; [X,stateSeq] = genhmm(hmm1); plotseq(X,stateSeq);
```

(To be repeated several times.)

Repeat with another model :

```
>> [X,stateSeq] = genhmm(hmm2);plotseq(X,stateSeq);
```

and re-iterate the experiment. Also re-iterate with model HMM3.

Questions :

1. How can you verify that a transition matrix is valid ?
2. What is the effect of the different transition matrices on the sequences obtained during the current experiment ? Hence, what is the role of the transition probabilities in the Markovian modeling framework ?
3. What would happen if we didn't have a final state ?
4. In the case of HMMs with plain Gaussian emission probabilities, what quantities should be present in the complete parameter set Θ that specifies a particular model ?

   If the model is ergodic with N states (including the initial and final), and represents data of dimension D, what is the total number of parameters in Θ ?
5. Which type of HMM (ergodic or left-right) would you use to model words ?

Answers :

1. In a transition matrix, the element of row i and column j specifies the probability to go from state i to state j. Hence, the values on row i specify the probability of all the possible transitions that start from state i. This set of transitions must be a complete set of discrete events. Hence, the terms of the i^{th} row of the matrix must sum up to 1. Similarly, the sum of all the elements of the matrix is equal to the number of states in the HMM.

(Answers continue on the next page...)

Answers (continued) :

2. *The transition matrix of HMM1 indicates that the probability of staying in a particular state is close to the probability of transiting to another state. Hence, it allows for frequent jumps from one state to any other state. The observation variable therefore frequently jumps from one "phoneme" to any other, forming sharply changing sequences like /a,i,a,y,y,i,a,y,y,$\cdots$/.*

   Alternately, the transition matrix of HMM2 specifies high probabilities of staying in a particular state. Hence, it allows for more "stable" sequences, like /a,a,a,y,y,y,i,i,i,i,i,y,y,$\cdots$/.

   Finally, the transition matrix of HMM4 also rules the order in which the states are browsed : the given probabilities force the observation variable to go through /a/, then to go through /i/, and finally to stay in /y/, e.g. /a,a,a,a,i,i,i,y,y,y,y,$\cdots$/.

   Hence, the role of the transition probabilities is to introduce a temporal (or spatial) structure in the modeling of random sequences.

   Furthermore, the obtained sequences have variable lengths : the transition probabilities implicitly model a variability in the duration of the sequences. As a matter of fact, different speakers or different speaking conditions introduce a variability in the phoneme or word durations. In this respect, HMMs are particularly well adapted to speech modeling.

3. *If we didn't have a final state, the observation variable would wander from state to state indefinitely, and the model would necessarily correspond to sequences of infinite length.*

4. *In the case of HMMs with Gaussian emission probabilities, the parameter set Θ comprises :*

   - *the transition probabilities a_{ij};*
   - *the parameters of the Gaussian densities characterizing each state, i.e. the means μ_i and the variances Σ_i.*

   The initial state distribution is sometimes modeled as an additional parameter instead of being represented in the transition matrix.

   In the case of an ergodic HMM with N emitting states and Gaussian emission probabilities, we have :

   - *$(N-2)\times(N-2)$ transitions, plus $(N-2)$ initial state probabilities and $(N-2)$ probabilities to go to the final state;*
   - *$(N-2)$ emitting states where each pdf is characterized by a D dimensional mean and a $D\times D$ covariance matrix.*

   Hence, in this case, the total number of parameters is $(N-2)\times(N+D\times(D+1))$. Note that this number grows exponentially with the number of states and the dimension of the data.

5. *Words are made of ordered sequences of phonemes : /h/ is followed by /e/ and then by /l/ in the word "hello". Each phoneme can in turn be considered as a particular random process (possibly Gaussian). This structure can be adequately modeled by a left-right HMM.*

   In "real world" speech recognition, the phoneme themselves are often modeled as left-right HMMs rather than plain Gaussian densities (e.g. to model separately the attack, then the stable part of the phoneme and finally the "end" of it). Words are then represented by large HMMs made of concatenations of smaller phonetic HMMs.

3 Pattern recognition with HMMs

3.1 Likelihood of a sequence given a HMM

In section 2, we have generated some stochastic observation sequences from various HMMs. Now, it is useful to study the reverse problem, namely : given a new observation sequence and a set of models, which model explains best the sequence, or in other terms which model gives the highest likelihood to the data ?

To solve this problem, it is necessary to compute $p(X|\Theta)$, i.e. the likelihood of an observation sequence given a model.

Useful formulas and definitions :

- *Probability of a state sequence*: the probability of a state sequence $Q = \{q_1, \cdots, q_T\}$ coming from a HMM with parameters Θ corresponds to the product of the transition probabilities from one state to the following :

$$P(Q|\Theta) = \prod_{t=1}^{T-1} a_{t,t+1} = a_{1,2} \cdot a_{2,3} \cdots a_{T-1,T}$$

- *Likelihood of an observation sequence given a state sequence*, or *likelihood of an observation sequence along a single path*: given an observation sequence $X = \{x_1, x_2, \cdots, x_T\}$ and a state sequence $Q = \{q_1, \cdots, q_T\}$ (of the same length) determined from a HMM with parameters Θ, the likelihood of X along the path Q is equal to :

$$p(X|Q,\Theta) = \prod_{i=1}^{T} p(x_i|q_i,\Theta) = b_1(x_1) \cdot b_2(x_2) \cdots b_T(x_T)$$

  i.e. it is the product of the emission probabilities computed along the considered path.

  In the previous lab, we had learned how to compute the likelihood of a single observation with respect to a Gaussian model. This method can be applied here, for each term x_i, if the states contain Gaussian pdfs.

- *Joint likelihood of an observation sequence X and a path Q* : it consists in the probability that X and Q occur simultaneously, $p(X, Q|\Theta)$, and decomposes into a product of the two quantities defined previously :

$$p(X,Q|\Theta) = p(X|Q,\Theta)P(Q|\Theta) \qquad \text{(Bayes)}$$

- *Likelihood of a sequence with respect to a HMM*: the likelihood of an observation sequence $X = \{x_1, x_2, \cdots, x_T\}$ with respect to a Hidden Markov Model with parameters Θ expands as follows :

$$p(X|\Theta) = \sum_{\text{every possible } Q} p(X,Q|\Theta)$$

  i.e. it is the sum of the joint likelihoods of the sequence over all possible state sequence allowed by the model.

- the *forward recursion*: in practice, the enumeration of every possible state sequence is infeasible. Nevertheless, $p(X|\Theta)$ can be computed in a recursive way by the *forward recursion*. This algorithm defines a forward variable $\alpha_t(i)$ corresponding to :

$$\alpha_t(i) = p(x_1, x_2, \cdots x_t, q^t = q_i|\Theta)$$

  i.e. $\alpha_t(i)$ is the probability of having observed the partial sequence $\{x_1, x_2, \cdots, x_t\}$ *and* being in the state i at time t (event denoted q_i^t in the course), given the parameters Θ. For a HMM with 5 states (where states 1 and N are the non-emitting initial and final states, and states $2 \cdots N-1$ are emitting), $\alpha_t(i)$ can be computed recursively as follows :

The Forward Recursion

1. **Initialization**

$$\alpha_1(i) = a_{1i} \cdot b_i(x_1), \quad 2 \leq i \leq N-1$$

where a_{1i} are the transitions from the initial non-emitting state to the emitting states with pdfs $b_{i,\, i=2\cdots N-1}(x)$. Note that $b_1(x)$ and $b_N x$ do not exist since they correspond to the non-emitting initial and final states.

2. **Recursion**

$$\alpha_{t+1}(j) = \left[\sum_{i=2}^{N-1} \alpha_t(i) \cdot a_{ij}\right] b_j(x_{t+1}), \qquad \begin{array}{c} 1 \leq t \leq T \\ 2 \leq j \leq N-1 \end{array}$$

3. **Termination**

$$p(X|\Theta) = \left[\sum_{i=2}^{N-1} \alpha_T(i) \cdot a_{iN}\right]$$

i.e. at the end of the observation sequence, sum the probabilities of the paths converging to the final state (state number N).

(For more detail about the forward procedure, refer to [?], chap.6.4.1).

This procedure raises a very important implementation issue. As a matter of fact, the computation of the α_t vector consists in products of a large number of values that are less than 1 (in general, *significantly* less than 1). Hence, after a few observations ($t \approx 10$), the values of α_t head exponentially to 0, and the floating point arithmetic precision is exceeded (even in the case of double precision arithmetics). Two solutions exist for that problem. One consists in scaling the values and undo the scaling at the end of the procedure: see [?] for more explanations. The other solution consists in using log-likelihoods and log-probabilities, and to compute $\log p(X|\Theta)$ instead of $p(X|\Theta)$.

Questions :

1. The following formula can be used to compute the log of a sum given the logs of the sum's arguments:

$$\log(a+b) = f(\log a, \log b) = \log a + \log\left(1 + e^{(\log b - \log a)}\right)$$

Demonstrate its validity.

Naturally, one has the choice between using $\log(a+b) = \log a + \log\left(1 + e^{(\log b - \log a)}\right)$ or $\log(a+b) = \log b + \log\left(1 + e^{(\log a - \log b)}\right)$, which are equivalent in theory. If $\log a > \log b$, which version leads to the most precise implementation ?

2. Express the log version of the forward recursion. (Don't fully develop the log of the sum in the recursion step, just call it "logsum": $\sum_{i=1}^{N} x_i \overset{\log}{\longmapsto} \mathrm{logsum}_{i=1}^{N}(\log x_i)$.) In addition to the arithmetic precision issues, what are the other computational advantages of the log version ?

Answers :

1. *Demonstration :*

$$a = e^{\log a} \quad ; \quad b = e^{\log b}$$

$$\begin{aligned} a + b &= e^{\log a} + e^{\log b} \\ &= e^{\log a}\left(1 + e^{(\log b - \log a)}\right) \end{aligned}$$

$$\log(a+b) = \log a + \log\left(1 + e^{(\log b - \log a)}\right) \quad QED.$$

The computation of the exponential overflows the double precision arithmetics for big values (≈ 700) earlier than for small values. Similarly, the implementations of the exponential operation are generally more precise for small values than for big values (since an error on the input term is exponentially amplified). Hence, if $\log a > \log b$, the first version ($\log(a+b) = \log a + \log\left(1 + e^{(\log b - \log a)}\right)$) is more precise since in this case $(\log b - \log a)$ is small. If $\log a < \log b$, it is better to swap the terms (i.e. to use the second version).

2. (a) **Initialization**

$$\alpha_1^{(log)}(i) = \log a_{1i} + \log b_i(x_1), \quad 2 \le i \le N-1$$

(b) **Recursion**

$$\alpha_{t+1}^{(log)}(j) = \left[logsum_{i=2}^{N-1}\left(\alpha_t^{(log)}(i) + \log a_{ij}\right)\right] + \log b_j(x_{t+1}), \qquad \begin{array}{l} 1 \le t \le T \\ 2 \le j \le N-1 \end{array}$$

(c) **Termination**

$$\log p(X|\Theta) = \left[logsum_{i=2}^{N-1}\left(\alpha_T^{(log)}(i) + \log a_{iN}\right)\right]$$

In addition to the precision issues, this version transforms the products into sums, which is more computationally efficient. Furthermore, if the emission probabilities are Gaussians, the computation of the log-likelihoods $\log(b_j(x_t))$ eliminates the computation of the Gaussians' exponential (see lab session 4).

These two points just show you that once the theoretic barrier is crossed in the study of a particular statistical model, the importance of the implementation issues must not be neglected.

3.2 Bayesian classification

Question :

The forward recursion allows us to compute the likelihood of an observation sequence with respect to a HMM. Hence, given a sequence of features, we are able to find the most likely generative model in a Maximum Likelihood sense. What additional quantities and assumptions do we need to perform a true Bayesian classification rather than a Maximum Likelihood classification of the sequences ?

Which additional condition makes the result of Bayesian classification equivalent to the result of ML classification ?

Answer :

To perform a Bayesian classification, we need the prior probabilities $P(\Theta_i|\Theta)$ of each model. In addition, we can assume that all the observation sequences are equi-probable :

$$\begin{aligned} P(\Theta_i|X,\Theta) &= \frac{p(X|\Theta_i,\Theta)P(\Theta_i|\Theta)}{P(X|\Theta)} \\ &\propto p(X|\Theta_i)P(\Theta_i) \end{aligned}$$

$P(\Theta_i)$ can be determined by counting the probability of occurrence of each model (word or phoneme) in a database covering the vocabulary to recognize (see lab session 4).

If every model has the same prior probability, then Bayesian classification becomes equivalent to ML classification.

3.3 Maximum Likelihood classification

In practice, for speech recognition, it is very often assumed that all the model priors are equal (i.e. that the words or phonemes to recognize have equal probabilities of occurring in the observed speech). Hence, the speech recognition task consists mostly in performing the Maximum Likelihood classification of acoustic feature sequences. For that purpose, we must have of a set of HMMs that model the acoustic sequences corresponding to a set of phonemes or a set of words. These models can be considered as "stochastic templates". Then, we associate a new sequence to the most likely generative model. This part is called the *decoding* of the acoustic feature sequences.

Experiment :

Classify the sequences $X_1, X2, \cdots X_6$, given in the file `data.mat`, in a maximum likelihood sense with respect to the six Markov models given in table 1. Use the function `logfwd` to compute the log-forward recursion expressed in the previous section. Store the results in a matrix (they will be used in the next section) and note them in the table below.

Example :

```
>> plot(X1(:,1),X1(:,2));
>> logProb(1,1) = logfwd(X1,hmm1)
>> logProb(1,2) = logfwd(X1,hmm2)
etc.
>> logProb(3,2) = logfwd(X3,hmm2)
etc.
```

Filling the `logProb` matrix can be done automatically with the help of loops :

```
>> for i=1:6,
  for j=1:6,
    stri = num2str(i);
    strj = num2str(j);
    eval([ 'logProb(' , stri , ',' , strj , ')=logfwd(X' , stri , ',hmm' , strj , ');' ]);
  end;
end;
>> logProb
```

Sequence	$\log p(X\|\Theta_1)$	$\log p(X\|\Theta_2)$	$\log p(X\|\Theta_3)$	$\log p(X\|\Theta_4)$	$\log p(X\|\Theta_5)$	$\log p(X\|\Theta_6)$	Most likely model
X1							
X2							
X3							
X4							
X5							
X6							

Answer :

$X_1 \rightarrow HMM1$, $X_2 \rightarrow HMM3$, $X_3 \rightarrow HMM5$, $X_4 \rightarrow HMM4$, $X_5 \rightarrow HMM6$, $X_6 \rightarrow HMM2$.

4 Optimal state sequence

Useful formulas and definitions:

In speech recognition and several other pattern recognition applications, it is useful to associate an "optimal" sequence of states to a sequence of observations, given the parameters of a model. For instance, in the case of speech recognition, knowing which frames of features "belong" to which state allows to locate the word boundaries across time. This is called the *alignment* of acoustic feature sequences.

A "reasonable" optimality criterion consists in choosing the state sequence (or *path*) that brings a maximum likelihood with respect to a given model. This sequence can be determined recursively via the *Viterbi algorithm*. This algorithm makes use of two variables:

- the *highest* likelihood $\delta_t(i)$ along a *single* path among all the paths ending in state i at time t:

$$\delta_t(i) = \max_{q_1,q_2,\cdots,q_{t-1}} p(q_1,q_2,\cdots,q_{t-1},q^t = q_i, x_1,x_2,\cdots x_t|\Theta)$$

- a variable $\psi_t(i)$ which allows to keep track of the "best path" ending in state i at time t:

$$\psi_t(i) = \underset{q_1,q_2,\cdots,q_{t-1}}{\arg\max}\; p(q_1,q_2,\cdots,q_{t-1},q^t = q_i, x_1,x_2,\cdots x_t|\Theta)$$

Note that these variables are vectors of $(N-2)$ elements, $(N-2)$ being the number of emitting states. With the help of these variables, the algorithm takes the following steps:

The Viterbi Algorithm

1. **Initialization**

$$\begin{aligned} \delta_1(i) &= a_{1i} \cdot b_i(x_1), \quad 2 \le i \le N-1 \\ \psi_1(i) &= 0 \end{aligned}$$

where, again, a_{1i} are the transitions from the initial non-emitting state to the emitting states with pdfs $b_{i,\,i=2\cdots N-1}(x)$, and where $b_1(x)$ and $b_N x$ do not exist since they correspond to the non-emitting initial and final states.

2. **Recursion**

$$\begin{aligned} \delta_{t+1}(j) &= \max_{2\le i\le N-1} [\delta_t(i)\cdot a_{ij}] \cdot b_j(x_{t+1}), \quad \begin{array}{l} 1 \le t \le T-1 \\ 2 \le j \le N-1 \end{array} \\ \psi_{t+1} &= \underset{2\le i\le N-1}{\arg\max} [\delta_t(i)\cdot a_{ij}], \quad \begin{array}{l} 1 \le t \le T-1 \\ 2 \le j \le N-1 \end{array} \end{aligned}$$

"Optimal policy is composed of optimal sub-policies": find the path that leads to a maximum likelihood considering the best likelihood at the previous step and the transitions from it; then multiply by the current likelihood given the current state. Hence, the best path is found by induction.

3. **Termination**

$$\begin{aligned} p^*(X|\Theta) &= \max_{2\le i\le N-1} [\delta_T(i)\cdot a_{iN}] \\ q_T^* &= \underset{2\le i\le N-1}{\arg\max} [\delta_T(i)\cdot a_{iN}] \end{aligned}$$

Find the best likelihood when the end of the observation sequence is reached, given that the final state is the non-emitting state N.

4. **Backtracking**

$$Q^* = \{q_1^*,\cdots,q_T^*\} \quad \text{so that} \quad q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \cdots, 1$$

Read (decode) the best sequence of states from the ψ_t vectors.

Hence, the Viterbi algorithm delivers *two* useful results, given an observation sequence $X = \{x_1, \cdots, x_T\}$ and a model Θ:

- the selection, among all the possible paths in the considered model, of the *best path* $Q^* = \{q_1^*, \cdots, q_T^*\}$, which corresponds to the state sequence giving a maximum of likelihood to the observation sequence X;
- the *likelihood along the best path*, $p(X, Q^*|\Theta) = p^*(X|\Theta)$. As opposed to the the forward procedure, where all the possible paths are considered, the Viterbi computes a likelihood along the best path only.

(For more detail about the Viterbi algorithm, refer to [?], chap.6.4.1).

Questions :

1. From an algorithmic point of view, what is the main difference between the computation of the δ variable in the Viterbi algorithm and that of the α variable in the forward procedure ?
2. Give the log version of the Viterbi algorithm.

Answers :

In this version, the logsum *operation (involving the computation of an exponential) is avoided, alleviating even further the computational load.*

(d) **Backtracking**

$$Q^* = \{q_1^*, \cdots, q_T^*\} \quad \textit{so that} \quad q_t^* = \psi_{t+1}(q_{t+1}^*) \quad t = T-1, T-2, \cdots, 1$$

(c) **Termination**

$$\begin{aligned} \log p^*(X|\Theta) &= \max_{2 \le i \le N-1} \left[\delta_T^{(log)}(i) + \log a_{iN} \right] \\ q_T^* &= \arg\max_{2 \le i \le N-1} \left[\delta_T^{(log)}(i) + \log a_{iN} \right] \end{aligned}$$

(b) **Recursion**

$$\begin{aligned} \delta_{t+1}^{(log)}(j) &= \max_{2 \le i \le N-1} \left[\delta_t^{(log)}(i) + \log a_{ij} \right] + \log b_j(x_{t+1}), \quad & \begin{array}{l} 1 \le t \le T-1 \\ 2 \le j \le N-1 \end{array} \\ \psi_{t+1} &= \arg\max_{2 \le i \le N-1} \left[\delta_t^{(log)}(i) + \log a_{ij} \right], \quad & \begin{array}{l} 1 \le t \le T-1 \\ 2 \le j \le N-1 \end{array} \end{aligned}$$

2. *(a)* **Initialization**

$$\begin{aligned} \delta_1^{(log)}(i) &= \log a_{1i} + \log b_i(x_1), \quad 2 \le i \le N-1 \\ \psi_1(i) &= 0 \end{aligned}$$

1. *The sums that were appearing in the computation of α become* max *operations in the computation of δ. Hence, the Viterbi procedure takes less computational power than the forward recursion.*

Experiments :

1. Use the function `logvit` to find the best path of the sequences $X_1, \cdots X_6$ with respect to the most likely model found in section 3.3 (i.e. X_1: HMM1, X_2: HMM3, X_3: HMM5, X_4: HMM4, X_5: HMM6 and X_6: HMM2). Compare with the state sequences $ST_1, \cdots ST_6$ originally used to generate $X_1, \cdots X_6$ (use the function `compseq`, which provides a view of the first dimension of the observations as a time series, and allows to compare the original alignment to the Viterbi solution).

2. Use the function `logvit` to compute the probabilities of the sequences $X_1, \cdots X_6$ along the best paths with respect to each model $\Theta_1, \cdots \Theta_6$. Note your results below. Compare with the log-likelihoods obtained in the section 3.3 with the forward procedure.

Examples :

1. Best paths and comparison with the original paths :
```
>> figure;
>> [STbest,bestProb] = logvit(X1,hmm1); compseq(X1,ST1,STbest);
>> [STbest,bestProb] = logvit(X2,hmm3); compseq(X2,ST2,STbest);
```
Repeat for the remaining sequences.

2. Probabilities along the best paths for all the models :
```
>> [STbest,bestProb(1,1)] = logvit(X1,hmm1);
>> [STbest,bestProb(1,2)] = logvit(X1,hmm2);
```
etc.
```
>> [STbest,bestProb(3,2)] = logvit(X3,hmm2);
```
etc. (You can also use loops here.)

To compare with the complete log-likelihood, issued by the forward recurrence :
```
>> diffProb = logProb - bestProb
```

Likelihoods along the best path :

Sequence	$\log p^*(X\|\Theta_1)$	$\log p^*(X\|\Theta_2)$	$\log p^*(X\|\Theta_3)$	$\log p^*(X\|\Theta_4)$	$\log p^*(X\|\Theta_5)$	$\log p^*(X\|\Theta_6)$	Most likely model
X1							
X2							
X3							
X4							
X5							
X6							

Difference between log-likelihoods and likelihoods along the best path :

Sequence	HMM1	HMM2	HMM3	HMM4	HMM5	HMM6
X1						
X2						
X3						
X4						
X5						
X6						

Question :

Is the likelihood along the best path a good approximation of the real likelihood of a sequence given a model ?

Answer :

The values found for both likelihoods differ within an acceptable error margin. Furthermore, using the best path likelihood does not, in most practical cases, modify the classification results. Finally, it alleviates further the computational load since it replaces the sum or the logsum by a max in the recursive part of the procedure. Hence, the likelihood along the best path can be considered as a good approximation of the true likelihood.

5 Training of HMMs

Decoding or aligning acoustic feature sequences requires the prior specification of the parameters of some HMMs. As explained in section 3.3, these models have the role of stochastic templates to which we compare the observations. But how to determine templates that represent efficiently the phonemes or the words that we want to model ? The solution is to estimate the parameters of the HMMs from a database containing observation sequences, in a supervised or an unsupervised way.

Questions :

In the previous lab session, we have learned how to estimate the parameters of Gaussian pdfs given a set of training data. Suppose that you have a database containing several utterances of the imaginary word /aiy/, and that you want to train a HMM for this word. Suppose also that this database comes with a *labeling* of the data, i.e. some data structures that tell you where are the phoneme boundaries for each instance of the word.

1. Which model architecture (ergodic or left-right) would you choose ? With how many states ? Justify your choice.
2. How would you compute the parameters of the proposed HMM ?
3. Suppose you didn't have the phonetic labeling (*unsupervised training*). Propose a recursive procedure to train the model, making use of one of the algorithms studied during the present session.

Answers :

1. *It can be assumed that the observation sequences associated with each distinct phoneme obey specific densities of probability. As in the previous lab, this means that the phonetic classes are assumed to be separable by Gaussian classifiers. Hence, the word /aiy/ can be assimilated to the result of drawing samples from the pdf* $\mathcal{N}_{/a/}$*, then transiting to* $\mathcal{N}_{/i/}$ *and drawing samples again, and finally transiting to* $\mathcal{N}_{/y/}$ *and drawing samples. It sounds therefore reasonable to model the word /aiy/ by a* left-right *HMM with* three *emitting states.*

2. *If we know the phonetic boundaries for each instance, we know to which state belongs each training observation, and we can give a label (/a/, /i/ or /y/) to each feature vector. Hence, we can use the mean and variance estimators studied in the previous lab to compute the parameters of the Gaussian density associated with each state (or each label).*

   By knowing the labels, we can also count the transitions from one state to the following (itself or another state). By dividing the transitions that start from a state by the total number of transitions from this state, we can determine the transition matrix.

3. *The Viterbi procedure allows to distribute some labels on a sequence of features. Hence, it is possible to perform unsupervised training in the following way :*

   (a) *Start with some arbitrary state sequences, which constitute an initial labeling. (The initial sequences are usually made of even distributions of phonetic labels along the length of each utterance.)*

   (b) *Update the model, relying on the current labeling.*

   (c) *Use the Viterbi algorithm to re-distribute some labels on the training examples.*

   (d) *If the new distribution of labels differs from the previous one, re-iterate (go to (b)). One can also stop when the evolution of the likelihood of the training data becomes asymptotic to a higher bound.*

   The principle of this algorithm is similar to the Viterbi-EM, used to train the Gaussians during the previous lab. Similarly, there exists a "soft" version, called the Baum-Welch *algorithm, where each state participates to the labeling of the feature frames (this version uses the forward recursion instead of the Viterbi). The Baum-Welch algorithm is an EM algorithm specifically adapted to the training of HMMs (see [?] for details), and is one of the most widely used training algorithms in "real world" speech recognition.*

After the lab...

This lab manual can be kept as additional course material. If you want to browse the experiments again, you can use the script :

```
>> lab2demo
```

which will automatically redo all the computation and plots for you.