VTT INFORMATION TECHNOLOGY

RESEARCH REPORT TTE5-2001-34

ImWell
T1SU00067

Hidden Markov Models – An Introduction in the Context of Biomedical Signal Interpretation Research

Version 1.01

8.1.2002

Mark van Gils

Version history

Version	Date	Author(s)	Reviewer	Description
0.1	2001-11-12	Mark van Gils		start first draft
0.8	2001-11-20	Mark van Gils	Juha Pärkkä	first full version
0.9	2001-12-03	Mark van Gils	Jyrki Lötjönen, Niilo Saranummi, Ilkka Korhonen	comments Juha Pärkkä incorporated
1.0	2001-12-17	Mark van Gils		reviewer comments incorporated
1.01	2002-1-8	Mark van Gils		Textual rephrasing for public distribution

Contact information

Mark van Gils
VTT Information Technology
P.O. Box 1206, FIN-33101 Tampere , Finland
Street Address: Sinitaival 6
Tel. +358 3 316 3342, fax +358 3 317 4102
Email: Mark.vanGils@vtt.fi
Web: http://www.vtt.fi/tte/

Last modified on 08.01.02

Abstract

Hidden Markov Models (HMMs) have been around for quite some time as a tool to classify data and study the mechanisms that produce those data. Traditionally HMMs have been used in speech recognition, but plenty of other application examples are available. Nowadays their application in BioIT is perhaps the most important field. This document aims to provide an introduction to the HMM concept and give an idea of their potential applicability in BSI (Biosignal Interpretation) research. A short introduction to HMMs is provided with the emphasis on practical use. An overview of available software and potential application areas completes this document.

Contents

List of symbols

A	transition matrix
a_{ij}	element of **A:** probability of moving from state *i* to state *j*
$\alpha_i(t)$	forward variable; probability of obtaining a partial observation sequence (from 1 to *t*) and being in state *i* at time *t*
B	observation probability distribution (*N* elements)
$b_j(k)$	probability that state *j* generates observation *k*
c_{jk}	linear combination coefficient for mixing Gaussians in state *j*
$G(\mathbf{o},\mathrm{m}_{jk},\mathbf{U}_{jk})$	Gaussian function with mean m_{jk} and covariance matrix $\mathbf{U}_{jk}$.
L	number of HMMs developed/trained in a certain application
λ	(hidden) Markov model
M	number of different observations/symbols that the model can produce
N	number of different states in a model
O	vector with observations
π	initial-state probability vector
q_t	state of the model at time *t*
t	time
V	set of (*M*) observations that can be generated by the model

1 Introduction

In many applications where we want to interpret recorded data we tend to make use of models. If we are able to make a valid model of the system that generated the data we are usually better able to make conclusions from the data at hand, perform classification and prediction tasks, and obtain a compact representation of the system. The trouble is however that we often are unable to uncover exactly what kind of model is generating the results that we observe, this is especially true if we are measuring for example from humans. The underlying system is then so complicated, and/or our knowledge about it so limited that we have to be content with using (too) simple models to represent it. Nevertheless, even a very simple model can be a powerful tool.

In the signal processing field, a wide variety of paradigms is known that allow us to make some kind of model of a system on the basis of measurements that we do (think about e.g., statistical models, artificial neural networks, AR modelling etc.). This document describes one of them: Hidden Markov Models (HMMs). These models are especially suited to model stochastic, non-stationary processes and study *sequences* of data. In essence it uses a stochastic finite-state machine as model, but we do not know necessarily how many states this machine has (the states are 'hidden'). The aim of this document is not to write an exhaustive introduction to the field; there are plenty of excellent other sources for that. For example, [Rabiner and Juang, 1993] is the standard reference and gives a thorough yet readable introduction in the context of speech recognition; [Cohen, 2002] gives a good, more practical, introduction in the context of biomedical signal processing. This document does aim to give a short introduction of what HMMs are, why they can be useful, especially in the context of BSI research, and what kind of tools currently exist to use them in practice. The latter issue is especially addressed by providing an overview of toolboxes available and an indication of useful WWW links.

2 Markov Processes (Chains)

As said, Markov modelling is about considering the system at hand as a finite state machine and considering the states through which it moves through over time. Before discussing about the 'real' hidden Markov models, it is good to consider models that deal with states that are 'visible' first.

Let's take as example a model for a patient suffering from diabetes (taken from http://www.pennmush.org/~alansz/courses/mdm-block/markov.html) , and let us assume that we can model his life using some states he is in. Once we have

diagnosed diabetes and decided to control it, we can model his state by, quite depressingly, using only three states, viz., a state of 'controlled diabetes', a state of 'end-stage renal disease' (ESRD), and finally a state of 'death'. We examine the patient's state once a year and then note in which of the three states the patient is in. He obviously can/will move from one state to the other, so he will move along a 'chain' of states. In Markov modelling we use probabilities to describe the chance that the patient moves from one state to another, and we can put those probabilities in a so-called *transition matrix*.

Let's assume we have the following probabilities:

- Each year, patients with controlled diabetes have an 85% chance of staying in that state, a 10% chance of moving to the ESRD state, and a 5% chance of dying.
- Each year, patients with ESRD have a 70% chance of continue being in the ESRD state and a 30% chance of dying.
- As we are not in a horror movie, we assume that patients who have died have a 100% chance of staying dead. Death is thus an "absorbing state" -- a state which, once entered, cannot be left.

On the basis of this we can make the following transition matrix;

$$\mathbf{A} = \{a_{ij}\} = \begin{pmatrix} 0.85 & 0.10 & 0.05 \\ 0.00 & 0.70 & 0.30 \\ 0.00 & 0.00 & 1.00 \end{pmatrix} \quad (1)$$

where we consider a_{ij} to be the probabilities of making the transition from state i to state j. Two important things should draw your attention here:

1. we use *probabilities* and thus are dealing with a ***stochastic*** approach, and thus are in general talking about models that are a based on a large number of data, and
2. the ***transition probabilities*** here are ***solely dependent on the current state***, i.e., there is no keeping track of history involved, so, it does not make any difference if someone has already been in 'controlled diabetes' for 25 years, or whether he has been there for only 1 year; the probability of moving to the ESRD state is in both cases 0.10. This is called a Markov chain of order 1 - there are chains that consider more states in the past as well, but we won't consider them here.

We can depict the whole chain in Figure 1.

Using this type of description we can answer quite many questions, for example,

- Imagine that we begin with 10,000 patients with controlled diabetes. In one year, how many patients will still have controlled diabetes? How many will have ESRD? How many will have died?
- In 10 years, how many patients are estimated will be in each state?
- What is the chance of a patient having the following observation sequence, **O** = {'controlled diabetes', 'controlled diabetes', 'controlled diabetes', 'death'}, i.e., exactly three years in 'controlled diabetes' and then suddenly dead?
- Or is it maybe more likely to have 2 years of controlled diabetes, then one year of ESRD and then dead?

You can experiment with the example at http://www.pennmush.org/cgi-bin/markov.cgi or you can use some basic calculations to find the answers.

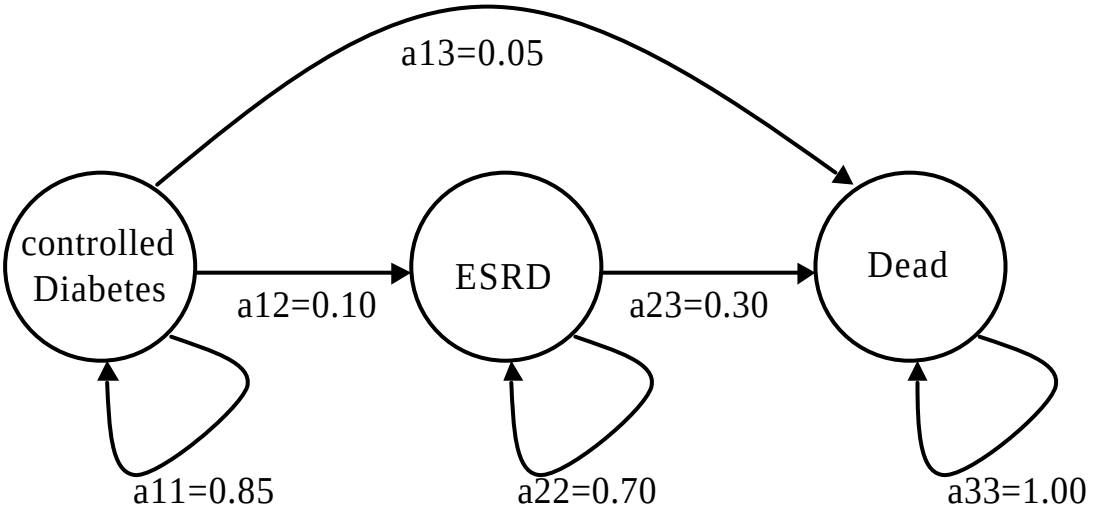

Figure 1: Markov chain of a three-state diabetes evolvement model.

As we will see later, those last 2 questions are important ones that we will use when trying to employ Markov models for interpretation/classification purposes. But, then we will use them the other way around: given our set of observations (recordings that we have done over some time) what is the most likely model that could have produced this observation sequence? The most-likely model is chosen from a set of models that we have made beforehand, and each model could have been labeled with some kind of 'class' (e.g., here classed associated with how the disease evolves, e.g., 'sudden death'-class vs. 'gradual decline'), chosing the most likely model is then similar to chosing the most likely class.

The example used here is of course a very simple one, but on the other hand quite representative. Markov modelling has its roots in the speech processing, but we nowadays see quite a lot of applications of them in processing other types of

signals as well, they are even used in completely other fields such as e.g., in health management or investigation of large molecule structures. In the health management application example we can attach costs or 'utility' to the different states and the instances at which they appear, and thus study the most economical path (when to intervene?). A much more realistic study using the 'diabetes application' can be found for example in [van Oss, Niessen et al., 2000].

The layout of the process chain in Figure 1 is also worth noting. The flow is going in one direction, from the left-to-the-right; there is no going back from ESRD to controlled diabetes and no rising from the dead either. The result is that the transition matrix in equation (1) has non-zero values only in its upper-triangle. Sometimes we have a pure sequential chain in which no 'skipping-over' is allowed (like the 'sudden-death' transition $a13$), which is even more simplified than the one described here. This is a quite common type of process and the model is used often. The most-obvious example is speech processing: moving through states (utterances of certain speech sounds) goes in a clear straightforward 'left-to-right' time-path (again, not considering horror movies and backwards spoken messages). The most general model however is the *ergodic model.* In such a model all states can always change to one of all other states, we thus have a transition matrix with, in principle, all elements having a non-zero value. An example for an ergodic model of the weather is given in Figure 2.

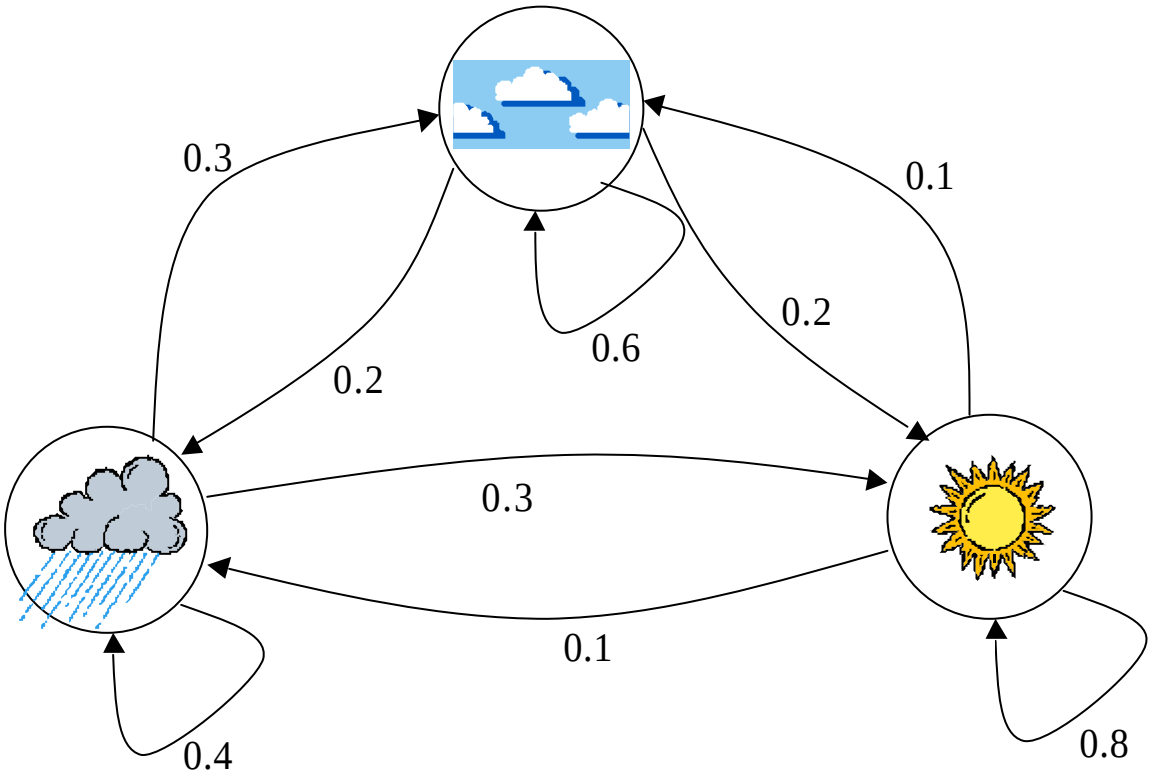

Figure 2: An example of an ergodic model for three states of different weather; rainy day, cloudy day, and sunny day.

Now, if we want to study the probability of being in a certain state, there is still one important piece of information missing. Namely, in which state does our process-chain actually start? In the diabetes example I mentioned implicitly that we always start in state 1, because 'we had diagnosed diabetes and decided to control it'. This is of course not always the case. In the weather example we might have started in any of the three cases, each with their own probabilities of occurring. In the diabetes example, if I would be interested in the diabetes states of the total population of a whole country I would need to take into account the the probability that someone has controlled diabetes or ESRD or is healthy (==no diabetes diagnosed). That would call for an extra, no-diabetes, state and expansion of our transition matrix to 4x4, but the main point is that we need to know the starting probabilities that anyone has controlled diabetes or is ESRD phase. We describe these probabilities by using a so-called *initial-state probability vector*, π. The probability of being in state *i* at time t=1 is the element *i* of the vector; π_i

If we take the weather example again and number 'rain' as state 1, 'cloudy' as state 2, and 'sunny' as state 3 and use days as time steps we can have for example for the weather in Bergen, Norway $\pi = [0.68, 0.22, 0.10]^T$, or if we are recording in Tampere we might have something like $\pi = [0.32, 0.50, 0.18]^T$ as initial-state probability vector[1]. We usually describe the model / Markov chain thus obtained as, λ, with

$$\lambda = \{\pi, \mathbf{A}\}. \quad (2)$$

Now, we usually are not directly recording 'states', but we typically have equipment that allows us to make recordings or observations. The output of a model is not the states, but rather the observations that we make. The observations are denoted by vector **O**. So, if we observe from day 1 until day *T* we have $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, ..., \mathbf{o}_T\}$, each $\mathbf{o}_i$ could be a vector containing precipitation and sunshine-hours measurements done during day *i*.

If one could say, in a *deterministic* fashion, that a rainy day (and only a rainy day) always brings us full-time rain, a sunny day always gives us 100% sunshine, and a cloudy day always gives us clouds in the sky (but no rain) for 24 hours, things would be simple. We have a one-to-one mapping of states to outputs/observations and we can unambiguously follow the model by looking at the observations; the states are immediately observable to us; they are not "hidden". Therefore this would be called an *observable Markov chain*.

In reality, we know very well that things are more complex and we can have some sunshine on a 'rainy day' and very well experience a few drops of rain on a 'sunny day'. However, the chances are more towards high precipitation in the former and

[1] The elements for raindays in the examples are actually based on climatological data, for the sunny and cloudy states I had to make some numbers up.

many hours of sunshine in the latter case. So, in reality each state has a certain *probability* of generating certain observations. This makes it trickier to figure out in which states we have been. Not only this, but, if we just have the observations to work with, who says that a model of three states underlies our weather observations, why not use four, or five, or two states? So, in reality even the number of states of the model can be hidden to us. These considerations will lead us to the ***Hidden*** *Markov Models*.

Another example of a Markov chain: sleep modelling, by defining a six-states Markov chain as proposed by Kemp [Kemp, 1987]. We have the states wake (W), REM-stage (REM), and sleep stages 1 (drowsiness) to 4. The model is depicted in Figure 3. The parameters $\{\pi, \mathbf{A}\}$ were estimated from 46 hypnograms. Signals generated by this model could not be distinguished from real hypnograms.

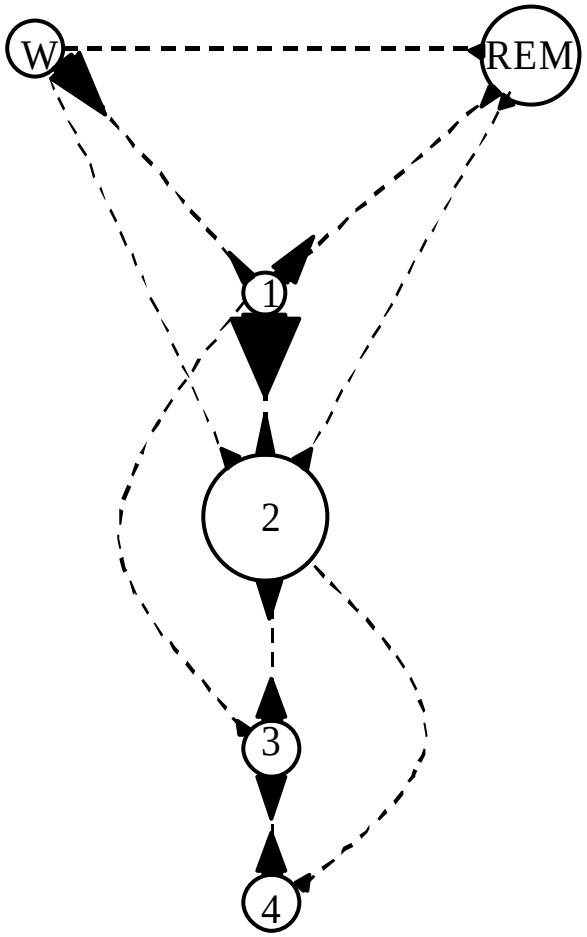

Figure 3: A model of a whole-night sleep structure as estimated from 46 hypnograms from 23 subjects. Circle areas are proportional to the time spent in the corresponding stage. Arrows indicate possible state transitions, arrow areas are (roughly) proportional to transition probabilities. There are a few transitions that are too small to be depicted. It can be seen e.g., that the most common stage is sleep stage 2, and in stage 1 the most likely transition is of that to stage 2. (after [Kemp, 1987])

3 Hidden Markov Models

As we could infer from the last section, an observable Markov model is usually insufficient to model real-life phenomena; we often cannot observe underlying states directly from the signals we record. We thus have to deal with the fact that a signal (observation) is a probabilistic function of the state, and a same observation may be generated from different states. Let's for the moment assume that our set of observations only contains discrete values, and they belong to a set V consisting of M distinct symbols $V = \{v_1, v_2, ... v_M\}$. Assuming our model consists of N different states, we denote the state we are in at time t as q_t. The probability that, at time t, state j generates observation/symbol v_k can be denoted as:

$$\begin{aligned} & b_j(k) = P(\mathbf{o}_t = v_k | q_t = j); \\ & j = 1,2,...,N; k = 1,2....,M \end{aligned} . \quad (3)$$

And, we now describe our model as

$$\begin{aligned} & l = \{p, \mathbf{A}, \mathbf{B}\} \\ & \text{with } \mathbf{B} = \{b_j\}. \end{aligned} \quad (4)$$

Given the knowledge we now have, we can think about three fundamental problems.

1. Given a sequence of observations $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, ..., \mathbf{o}_T\}$ and a model λ, compute $P(\mathbf{O} | \lambda)$, i.e., the probability that we record the sequence of data $\mathbf{O}$ given that the underlying model is λ.

2. Given a sequence of observations $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, ..., \mathbf{o}_T\}$ and a model λ, which sequence of states $q = \{q_1, q_2, q_3, ..., q_T\}$best 'explains' this sequence of observations, i.e., what is the most likely path that was followed through the model states during the measurement?

3. How do we find the right parameters for $\lambda = \{\pi, \mathbf{A}, \mathbf{B}\}$ so that we have an optimal $P(\mathbf{O} | \lambda)$?

Problem 1: is an extremely useful one, it allows us to consider the following: if we have different models available to choose from, which one fits the data best? If each of the available ones can be associated with a certain 'class' we can in fact make a **classifier** on the basis of this.

Problem 2: how do we find the 'correct' state sequence? It should be clear by now that, unless the problem is trivial, this cannot be found unambiguously. We have to use some kind of optimality criterion that we try to meet as well as possible to

find a most likely state sequence. This can lead us to **insight in the model**, find optimal state sequences for **(time-related) pattern recognition tasks**, average statistics of states and transitions (like e.g., the surface areas of circles and arrows in Figure 3) etc.

Problem 3: a very crucial problem; we need some initial observations to make an attempt at finding a usable model. This really is a matter of **training**; adapt model parameters so that the model represents real phenomena as well as possible.

It depends a bit on the application on how we want to investigate or classify data. In speech recognition for example we make/train a different model for each different word that should be recognized (sequence of syllables/phoneme-produced-sounds). So, each separate word has some different 'best fitting HMM' associated with it. When the 'recognizing task' is started we look which HMM model fits best to the first segment[2] of data and thus we get the 'word' that best fits the given segment of data. In the next segment we again choose the best fitting HMM and thus find the next word etc.

Alternatively, of course we could also have an application, like e.g., the Sleep-model of Figure 3, in which one single HMM model covers the whole data and then we are actually more interested in seeing which states are visited - to make some sleep-stage classifier for example. For many of our biomedical signal interpretation applications the latter approach is probably most relevant, but you could also think of applications with the first approach (e.g., classify separate ECG cycles, or segments of EEG). The first approach is typically used for 'high-volume' / raw-data applications - and much data is what you need to apply it, because you'll have to train many HMMs.

3.1 Solving the Three Basic Problems for HMMs

Let's take a look at how these three problems we found earlier are usually dealt with. The explanations here are deliberately kept short; the full details are excellently described in [Rabiner and Juang, 1993; Cohen 2002].

3.1.1 Evaluation of a model (classification/recognition)

Let's assume we have somehow managed to create (by solving Problem 3) L HMM models, we thus have a set of λ's denote by λi with i = 1...L. We will classify our signal at hand as being produced by model j by using

$$j = \arg\max\{P(\mathbf{O}|\lambda_i)\}. \quad (5)$$

We can do this very straightforwardly by enumerating through all possible N^T possible state sequences and their probably generated observations and see what

[2] for reasons of simplicity we assume we were somehow able to segment the data - how, that's another story.

the joint probabilities are (see [Rabiner and Juang, 1993, pp. 334-335] for the details). This is not a good idea. For example, for 100 observations and 5 states, this would lead to approximately 10^{72} computations, we obviously need something more efficient. This is done by 'dynamic programming' and using a so-called ***forward variable***, which is defined as the probability of getting a partial observation sequence (starting at time 1 and ending at time t) and being in state i at time t (Figure 4);

$$\alpha_t(i) = P(\mathbf{o}_1, \mathbf{o}_2, ,\ldots, \mathbf{o}_t, q_t = i \,|\, \lambda) . \tag{6}$$

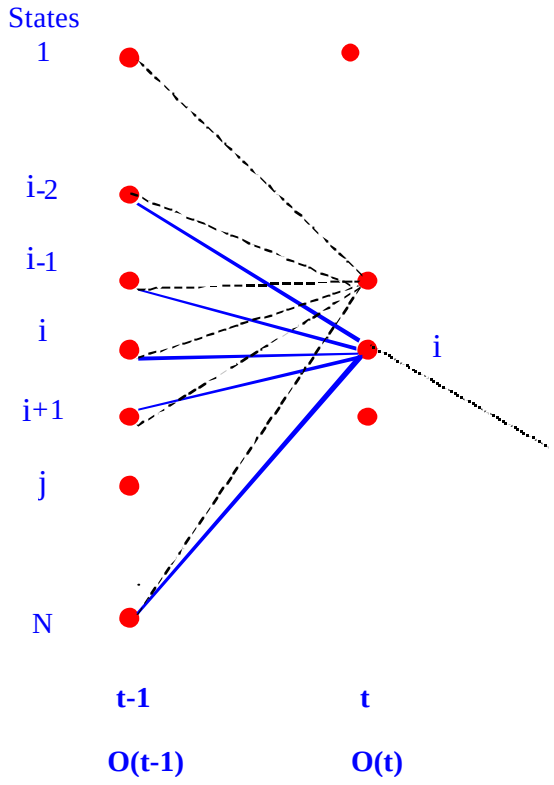

Figure 4: The forward-variable, $a_t(i)$, is calculated from the solid blue line path (some other possible paths that can be taken in the model, but are not in part of $a_t(i)$ are represented by dashed lines).

It can easily be shown that the forward variable may be calculated iteratively; use the initial condition

$$\alpha_1(i) = \pi_i b_i(\mathbf{O}_1) \;; \quad i = 1,2,..,N , \tag{7}$$

an induction rule

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{i,j}\right] b_j(\mathbf{o}_{t+1}); \quad j = 1,2,..,N; \quad t = 1,2,...T-1 \tag{8}$$

then we can reach a final result:

$$P(\mathbf{O}|\lambda) = \sum_{j=1}^{N} \alpha_T(j) \tag{9}$$

Equations (7) to (9) form an efficient algorithm, the ***forward procedure***, for output probability calculation.
We can do the same exercise, by using a so-called backward variable, β that considers the partial sequence obtained from t+1 to T and can be used to do a similar induction process, but then backwards. Both methods give the exact output probability. We can speed this process up still if we are happy with an approximation of the output probability. We can do this by considering the maximal local probability only rather than the sum over all possible probabilities (i.e., use the most likely path of problem 2). As it turns out, the most likely path really dominates the sum very much, and the error you make is small in comparison with the large saving in computation time.

3.1.2 Finding the most likely state sequence

As mentioned, obtaining an idea of what states were visited while we doing our observations can be useful for a number of applications. Since the states are hidden, we somehow have to make an estimation of what was most likely. We do this by employing the so-called ***Viterbi algorithm***. We would like to maximize the probability of experiencing state sequence $\mathbf{Q}$ given our observations and model at hand:

$$P(\mathbf{Q}|\mathbf{O},\lambda) = \frac{P(\mathbf{Q},\mathbf{O}|\lambda)}{P(\mathbf{O}|\lambda)}. \tag{10}$$

Again the details of this algorithm can be consulted from the two main references. The algorithm uses both the forward and backward variables introduced earlier. It employs the maximum probabilities rather than summing over all possible probabilities as mentioned in the preceding section, but otherwise has strong similarities with the forward procedure. Potential implementation difficulties exist due to underflow, as the probabilities and their multiplications can lead to very small numbers, therefore some scaling methods are used. Also, to reduce the dynamic range of the calculation and use additions instead of multiplications a version that uses the logarithmic values of the parameters is often used.

3.1.3 Finding the model parameters (training)

Given a set of data (training database), how do find a suitable HMM to fit to it? The maybe best known method for this is called the ***Baum-Welch algorithm*** (aka ***Expectation Maximization*** method), but there are others, e.g., gradient-based techniques. In a sense, the training is very much 'neural network like'; we start with an initial guess of parameters $\lambda_0 = \{\pi_0, \mathbf{A}_0, \mathbf{B}_0\}$ and then use re-estimation equations to update the parameters to get a model that suits the training data better in the sense of output probability (the one of problem 1). Based on the initial model we can make an expectation of the number of times each state or transition will be used (like e.g. the model in Figure 3 shows). We update the model parameters to maximize the likelihood that these numbers occur. After several (=not many; maybe 10-20) iterations we converge to a model with the maximum output probability (it is in fact a maximum likelihood estimator) And, similar to many neural nets training we also here run the risk that our model is sub-optimal because we ran into a local maximum (of which there are many).

Again, please read through [Rabiner and Juang, 1993] for the details, it does not make sense to copy them here. As opposed to gradient techniques, the Baum-Welch algorithm guarantees monotonical increase in the model fit for each iteration. A point of critique that exists with respect to the Baum-Welch method is that its results are dependent on the initialization, and for that reason alternative methods are being proposed as well. Especially a sensible choice of the elements of $\mathbf{B}_0$ is important, the exact choices for $\mathbf{A}_0$ and π_0 are less critical. A way to deal in practice with this is to repeat the training many times with different initializations and study the results of the repeated training sessions.

3.2 Continuous-Density HMMs (CD-HMMs)

When we discussed the outputs/observations that we could record from the system that we are modelling we stated that we limited ourselves to a set of outputs, V with $V = \{v_1, v_2, \ldots v_M\}$. This is fine if we can readily divide our observations into discrete classes, but what do we do if we record real-valued ('continuous') data? A way of dealing with this, and a quite often used one too, is by quantizing our real-value measurements first into 'prototypes' and then use the quantized data as observations. This requires some vector quantizer algorithm, such as LVQ, to pre-process our data first. We have to hope that our quantizer does an adequate job of dividing our data into good prototypes, this usually thus requires another training task; the training of the vector quantizer.

Another way to get around the problem is by employing so-called Continuous-Density HMMs (CD-HMMs) that allow us to use features that are continuous. A CD-HMM uses probability density functions (PDFs) rather than the discrete output probabilities that were in our vector **B**. The way we go about this is by considering the PDF of the observation vector **o**, while being in state j, denoted

$b_j(\mathbf{o})$, as a being in the shape of a linear combination of Gaussian[3] functions. In other words, the PDF is a mixture of Gaussians:

$$b_j(\mathbf{o}) = \sum_{k=1}^{M} c_{jk} G(\mathbf{o}, \mathbf{m}_{jk}, \mathbf{U}_{jk}); \quad j = 1,2,..,N$$
$$\text{with } \sum_{k=1}^{M} c_{jk} = 1; \quad c_{jk} \geq 0, \; k = 1,2,..,M \tag{11}$$

With function G being a Gaussian function with mean m_{jk} and covariance matrix $\mathbf{U}_{jk}$. In principle each state PDF could be described by a different number of Gaussians, but usually one uses the same number of Gaussians for all of them. The linear combination coefficients c_{jk}, mean m_{jk} and covariance matrix $\mathbf{U}_{jk}$ can be estimated using a modified version of the Baum-Welch training algorithm. This is done on the basis of the realization that a state with an M-mixture-density PDF is actually equivalent to M 'sub'states with single densities. In general the CD-HMM gives more accurate results than its discrete counterpart, however, its training is also more complex and it requires a larger training database.

The initial choice of the conditions here is important, much more so than in the discrete case; the CD-HMM is very sensitive to initial conditions of the observation PDFs. There are some techniques to tackle this problem.

The first one first trains a DD-HMM (using VQ-preprocessed data); observations are matched with states (using the Viterbi algorithm) and subsequent state clusters are formed. The cluster of the ith state is then further divided into M subclusters; each one assumed to be Gaussian and have a mean and covariance as estimated from the data. The relative number of observations in the subclusters can then be used as estimation for the linear combination coefficients $c_{jk.}$

Another technique uses educated guessing. For example, if we have a left-to-right HMM, we know that the first part of our data will originate from the first states, and we know that at the end of the recording we are supposed to be in the last state, and we can maybe linearly interpolate in between. Sometimes we know from the nature of the process what states are most likely to be most frequented. We can thus make clusters and then proceed.

3.3 Other types of HMMs

There are many other types and flavours of HMMs that are tailor-made to fit better to certain applications or to overcome certain problems. To name but a few:

- HMMs in which the observations are not generated by states themselves but rather by the transition from one state to the other, these are called *Mealy*

[3] In fact any kind of 'log-concave' function will do, but usually Gaussians are used.

HMMs. This type of HMM is also used in speech recognition where transitions between states are very important as well to recognize.

- *Auto-regressive HMMs* (AR-HMMs). Here the continuous PDF is considered to be formed as an autoregressive process, and it is assumed that the *k*th observation can be approximated as a linear combination of previous observations. In that case, and if *T* is large, we can get a good approximation of the PDFs (see e.g., [Rabiner and Juang, 1993]).
- *Variable duration HMMs.* A standard HMM describes the time that a certain state is maintained for a continuous time as an exponentially declining function. However, this may not be appropriate for the phenomenon under study, and we may want to introduce an explicit duration probability for the states. When we enter a certain state we draw a number from our predefined duration probability, and stay in that for the corresponding amount of time. Only after that time has elapsed we again start looking into transitions to other states.

4 Applying HMMs

As mentioned, the main applications of HMMs lie in recognition; train *L* different HMMs representing different classes of data, subsequently provide new data, see which of the models suits best and perform the classification on the basis of that. This can also be used in segmentation tasks. The other application area concentrates on studying the properties of models obtained and learn something about the underlying states and their transitions.

In essence these two views are quite common in many other (biosignal) interpretation paradigms. There are people who use ANNs / statistical models / expert systems as a given tool to provide some output to be used in a certain (classification) task and people that study the actual contents of the model obtained during the learning phase. Probably the application of HMMs has strongest resemblance to that of ANNs (indeed many a neural-network conference and journal incorporates sections devoted to HMMs). Both implement a version of associative processing rather than deductive processing, both depend very much on learning and only work by the grace of having lots of workable data available. Perhaps the biggest difference between them is that HMMs have a more " system-*output*" approach to studying a system than ANNs have (that tend to work more with *input-output* associations); this suggests that there combination would be useful. HMMs are especially suited for sequence and time-series processing, whereas ANNs are not that great in this. Furthermore, the parameters of a HMM are very easy interpretable, whereas with many ANNs the interpretation of parameters (neural network weights) requires some experience. Both require some careful training (a.o. due to presence of local minima in error surfaces), but with

HMMs the training happens several orders of magnitude faster than with ANNs, both require techniques like cross-validation to assess their performance.

A very comprehensive overview of papers on the subject of applying HMMs can be found at http://www-sig.enst.fr/~cappe/docs/hmmbib.html. There are obvious applications in speech recognition and communication, but you can also see applications like 'climatology' and 'econometrics', and even 'recognition of international crises' [Schrodt, 1998]; areas which really concern the exploration of the models' properties.

Perhaps the largest activity (in terms of number of studies as well as money at stake) in HMM applications is in the 'bio-it' area (gene recognition, DNA sequencing, aligning etc).

4.1 Software Implementations

Several HMM-based software products are freely available. They are listed in Appendix A. Most of them, as a consequence of the roots of the method, are targeted at people using UNIX/Linux and working with speech recognition. There are several Matlab toolboxes, and then there are some commercial packages for use on Windows platforms, but those latter tend to be tailor-made for BioIT applications. For our group probably the Matlab toolboxes are the most useful.

Perhaps the most comprehensive one is freely available HTK. It is a set of C library modules and tools that was initially used for speech recognition research (using continuous density HMMs), but it has users in other fields as well. The source needs to be compiled (known to run on UNIX / Linuxes, and there are reports that it runs under WinNT as well) and run via command-line calls.

Some Matlab toolboxes are available. Perhaps the one of the most useful ones can be found at http://www.cs.berkeley.edu/~murphyk/Bayes/hmm.html. It offers a lot flexibility and supports different flavours of HMMs.

University of Maryland Hidden Markov Model (HMM). C-source implementation of Forward-Backward, Viterbi, and Baum-Welch algorithms. The software has been compiled and tested on UNIX platforms (sun solaris, dec osf and linux) and PC NT running a GNU package add-on.

5 Conclusions

We have seen that HMMs provide a tool for performing classification tasks as well as, and perhaps more interesting, a tool to study/model different processes. They bear quite some similarities to ANNs, but they have their specific strengths, especially when it comes to processing sequences of data. Data generated from many different applications might be well suited for exploration with HMMs.

Combination of HMMs with ANNs may give more powerful systems in future work. The bulk of the HMM software available seems to be traditionally UNIX-speech recognition directed, or, more recent, and especially fast coming up, directed to the field of BioIT.

Without trying to recommend the *technique du jour* to process whatever data we happen to have in store, it is the author's opinion that HMMs do possess enough unique and relevant merits to explore them further in real applications. A suggestion would be to use the available Matlab toolboxes to explore some existing databases one has collected earlier and get some experience in HMM usage. After that, they could readily be used in future projects.

Appendix A: Toolboxes

HTK
http://htk.eng.cam.ac.uk/docs/docs.shtml (accessed 16.11.2001)
C source code, originating from the field of at speech processing. Needs to be compiled and used as part of a larger infrastructure. Requires Unix-system or Linux (and is reported run on Windows NT as well).

HMM
http://www.cs.berkeley.edu/~murphyk/Bayes/hmm.html (accessed 16.11.2001)
Matlab 5 toolbox
Pretty comprehensive set of HMM matlab routines. HMM with discrete outputs, mixtures of Gaussians are supported. Documentation is quite sparse though.

DHMM - HMM
http://www.gatsby.ucl.ac.uk/~zoubin/software.html (accessed 16.11.2001)
Matlab toolbox very similar to the HMM toolbox, but slightly more limited.

UMHMM
http://www.cfar.umd.edu/~kanungo/software/software.html (accessed 19.11.2001)
University of Maryland Hidden Markov Model (HMM) C-source: Implementation of Forward-Backward, Viterbi, and Baum-Welch algorithms. The software has been compiled and tested on UNIX platforms (sun solaris, dec osf and linux) and PC NT running the GNU package from Cygnus (has gcc, sh, etc.). Quite similar to HTK, but maybe less well known.

Appendix B: Useful Web links

http://www.pennmush.org/~alansz/courses/mdm-block/markov.html (accessed 13.11.2001): Simple explanation of Markov chains by using the diabetes example, has a link to a model that you can use interactively to explore how states evolve over time.

http://www.marypat.org/stuff/random/markov.html (accessed 19.11.2001)
Collection of references, bookmarks etc. Useful (although quite many dead links).

http://www-sig.enst.fr/~cappe/docs/hmmbib.html (accessed 14.11.2001) "Ten years of HMMs". Very extensive and well-organized literature list of articles that describe applications of HMMs and were published 1989 and end 2000.

http://www.paracel.com/publications/hmm_white_paper.html (accessed 19.11.2001) White paper on HMM and Bio IT (biased towards GeneMatch, but useful nevertheless)

References[4]

[Cohen, 2002] Arnon Cohen, *Signal Processing*, 2nd edition, CRC press. To appear May 2002.

[Kemp, 1987] Bob Kemp, *Model-based monitoring of human sleep stages* PhD. Thesis Enschede, The Netherlands, 1987.

[van Oss, Niessen et al., 2000] Nicole van Oss, Louis Niessen, Henk Bilo, Anton Casparie, Ben van Hout, "Diabetes nephropathy in the Netherlands: a cost effectiveness analysis of national clinical guidelines", *Health Policy*, 51 (2000) 135–147.

[Rabiner and Juang, 1993] Lawrence Rabiner and Bing-Hwang Juang, *Fundamentals of Speech Recognition*, Prentice Hall Signal Processing Series (series Ed. Alan V. Oppenheim), Prentice Hall, Englewood Cliffs NJ, (1993) 321-389.

[Schrodt, 1998] Philip A. Schrodt, "Pattern Recognition of International Crises using Hidden Markov Models." In: *Non-linear Models and Methods in Political Science*, University of Michigan Press, Ann Arbor (1998).

[Silipo et al., 1999] Rosaria Silipo, Gustavo Deco, Rossano Vergassola, and Celio Gremigni, "A Characterization of HRV's Nonlinear Hidden Dynamics by Means of Markov Models", *IEEE Transactions on Biomedical Engineering*, 46 (1999) 978-986.

[4] This is a short list; real large reference lists can be found at the relevant URLs listed in Appendix B.