MARKOV MODELS
AND THEIR APPLICATION IN SPEECH RECOGNITION

Paul van Alphen & Dick R.van Bergem

1 INTRODUCTION

In current speech recognition research the use of hidden Markov models becomes more and more successful. However, for those who know little or nothing about (hidden) Markov models, no accessible introduction exists to our knowledge. What is a Markov model? How can Markov models be used as recognizers? Why is a hidden Markov model hidden? In this article an attempt has been made to answer these questions step by step, using numerical examples to clarify the algorithms that are presented.
The text has been divided in two major parts. In the first part the theory of Markov models is presented. To increase the readability of this part no references are given; instead we have included a list of recommended literature. In the second part the application of hidden Markov models to speech is demonstrated in a simple recognition task with ten Dutch digits.

2 MARKOV MODELS

2.1 Some remarks about probabilities

For a proper understanding of Markov models two basic manipulations with probabilities are of major importance: multiplying and adding. We will give an example to illustrate the difference between addition and multiplication of probabilities. How do we calculate the probability that the total score, resulting from throwing two unbiased dice, is exactly 8? There are $6^2 = 36$ possible combinations of dice numbers that may appear. Each of these combinations has a probability of $1/6 \cdot 1/6 = 1/36$ (=multiplication). There are 5 combinations that lead to a total score of 8: (2,6), (3,5), (4,4), (5,3), (6,2). Therefore the required probability is $1/36 + 1/36 + 1/36 + 1/36 + 1/36 = 5/36$ (=addition). This means that 5 *paths* out of the possible 36 lead to the desired result, a notion that is frequently used in the rest of this paper.

2.2 Introduction to Markov processes

Suppose you want to make a contribution to a cleaner environment. Instead of going to work with your car every day, you decide to take the train every now and then based on the following scheme. For the first day you toss a (fair) coin. If head appears you take the train, else you go by car. For each of the following days you will take the train if you went by car the previous day, unless a "5" or a "6" appears when you toss a die (a probability of 2/6) in which case you drive again to work. If, however, you went by train the previous day you will drive your car, unless a "6" appears when you toss a die (a probability of 1/6) in which case you will take the train again. This scheme can be modelled in the Markov chain of figure 1.

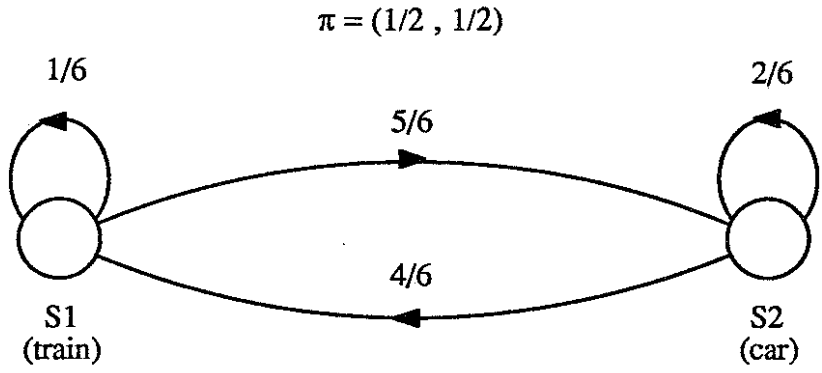

Figure 1. Example of a Markov chain applied to a 'travelling scheme' (see text).

A Markov chain is a stochastic process, whose outcome is a sequence of T observations O(1), O(2) ..... O(T) that satisfy the following assumptions:

1. Each observation belongs to a finite set of N states { S_1, S_2, ... , S_N }. If the outcome at time t is S_i, then we say that the system is in state S_i at that time.

2. Any observation depends only upon the immediately preceding observation and not upon any other previous observation. For each pair of states { S_i, S_j } a_{ij} denotes the probability that S_j occurs immediately after S_i occurs.

The numbers a_{ij}, called the *state transition probabilities* can be arranged in a matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & \dots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \dots & \dots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \dots & \dots & a_{3N} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & a_{N3} & \dots & \dots & a_{NN} \end{pmatrix}$$

called the transition matrix, where N is the total number of states.
The *initial state probability distribution*, i.e. the probability distribution when the process begins (t=1) can be arranged in a vector

$$\pi = (\pi_1, \pi_2, \pi_3, ..., \pi_N)$$

In our example there are two states S_1 and S_2 (N=2), representing the way of travelling to work at a certain day. This depends only on the way of travelling on the previous day (the previous state) and the transition probabilities, which are determined by tossing a die. The transition matrix is:

$$A = \begin{pmatrix} 1/6 & 5/6 \\ 4/6 & 2/6 \end{pmatrix}$$

The initial state distribution in our example is determined by the coin toss:

$$\pi = (1/2, 1/2)$$

What is the probability that you will drive your car on the fourth day (T=4) of your new way of life? To answer this question you must follow each possible state sequence

ending in state S_2, calculate the probability of each path and finally add the probabilities of all these paths. You can calculate these probabilities efficiently using an algorithm called the *forward procedure*. For this procedure we define the probability $\alpha_i(t)$ as the probability of being in state S_i at time t. Thus the vector $\alpha(t)$ denotes the probability distribution at time t. For our example we find:

$$\begin{aligned} \alpha_1(1) &= \pi_1 \\ &= 1/2 \end{aligned}$$

$$\begin{aligned} \alpha_2(1) &= \pi_2 \\ &= 1/2 \end{aligned}$$

because the probability of going by train or by car (being in state S_1 or S_2) on the first day is determined by a coin tossing. On the second day you may be either in state S_1 or in state S_2. Suppose you are in state S_1 on the second day. Then you may have come from state S_1 or from state S_2. The probability that you were in these states at t=1 you just calculated in $\alpha(1)$. These probabilities must be multiplied with the appropriate transition probabilities and the sum of the resulting probabilities is the probability of being in state S_1 on the second day. In the same way the probability of being in state S_2 on the second day can be calculated. For our example the calculations are as follows:

$$\begin{aligned} \alpha_1(2) &= \alpha_1(1) \cdot a_{11} + \alpha_2(1) \cdot a_{21} \\ &= 1/2 \cdot 1/6 + 1/2 \cdot 4/6 \\ &= 5/12 \end{aligned}$$

$$\begin{aligned} \alpha_2(2) &= \alpha_1(1) \cdot a_{12} + \alpha_2(1) \cdot a_{22} \\ &= 1/2 \cdot 5/6 + 1/2 \cdot 2/6 \\ &= 7/12 \end{aligned}$$

In general we can recursively calculate $\alpha(t)$ with the aid of $\alpha(t\text{-}1)$ using the formula:

$$\alpha_j(t) = \sum_{i=1}^{N} \alpha_i(t\text{-}1) \cdot a_{ij} \qquad (1)$$

For our example the calculations for the third and fourth day proceed as follows:

$$\begin{aligned} \alpha_1(3) &= \alpha_1(2) \cdot a_{11} + \alpha_2(2) \cdot a_{21} \\ &= 5/12 \cdot 1/6 + 7/12 \cdot 4/6 \\ &= 33/72 \end{aligned}$$

$$\begin{aligned} \alpha_2(3) &= \alpha_1(2) \cdot a_{12} + \alpha_2(2) \cdot a_{22} \\ &= 5/12 \cdot 5/6 + 7/12 \cdot 2/6 \\ &= 39/72 \end{aligned}$$

$$\begin{aligned} \alpha_1(4) &= \alpha_1(3) \cdot a_{11} + \alpha_2(3) \cdot a_{21} \\ &= 33/72 \cdot 1/6 + 39/72 \cdot 4/6 \\ &= 189/432 \end{aligned}$$

$$\begin{aligned} \alpha_2(4) &= \alpha_1(3) \cdot a_{12} + \alpha_2(3) \cdot a_{22} \\ &= 33/72 \cdot 5/6 + 39/72 \cdot 2/6 \\ &= 243/432 \end{aligned}$$

So the probability of driving your car (state S_2) on the fourth day is 243/432 and the probability of going by train (state S_1) is 189/432.
Note that since $\alpha(t)$ is a probability vector the sum of its components is always 1. Those who are familiar with matrix algebra will also notice that the vector $\alpha(t)$ can be defined as:

$$\alpha(t) = \pi \cdot A^{t-1} \tag{2}$$

in which A^{t-1} is the (t-1)-th power of A.

Another interesting question is: what is the most likely path through the states or in other words what is the most likely observation sequence? In our example there are two states S_1 (train) and S_2 (car). If we look at the first four days the number of possible observation sequences O(1), O(2), O(3), O(4) is 2^4. Which of these 16 sequences is most likely to occur? This question can be solved with the *Viterbi algorithm* which finds the single best path with the highest probability based on a dynamic programming method. We define $\delta_j(t)$ as the best score (highest probability) at time t along a single path that ends in state S_j. This score can be recursively calculated with the formula:

$$\delta_j(t) = \max_{1 \le i \le N} \left(\delta_i(t-1) \cdot a_{ij} \right) \tag{3}$$

The Viterbi algorithm is similar to the forward procedure. However, a maximization over previous states is used instead of the summing procedure used in the forward calculation. The optimal path can be found by backtracking if we store the *index i* of the $\delta_i(t-1)$ that maximizes $\delta_j(t)$ (according to formula (3)) in a vector. This vector $\psi_j(t)$ is simply a pointer to the 'best' preceding state S_i.

$$\psi_j(t) = \operatorname{argmax}_{1 \le i \le N} \left(\delta_i(t-1) \cdot a_{ij} \right) \tag{4}$$

The calculations for our example will clarify this backtracking strategy (the highest probability $\delta_i(t-1) \cdot a_{ij}$ has been underlined for each state in each of the four time steps).

$\delta_1(1) = \pi_1$
$= 1/2$

$\delta_2(1) = \pi_2$
$= 1/2$

$\delta_1(2) = \max [\, \delta_1(1) \cdot a_{11}, \delta_2(1) \cdot a_{21} \,]$ $\qquad \psi_1(2) = 2$
$= \max [\, 1/2 \cdot 1/6, \underline{1/2 \cdot 4/6} \,]$
$= 4/12$

$\delta_2(2) = \max [\, \delta_1(1) \cdot a_{12}, \delta_2(1) \cdot a_{22} \,]$ $\qquad \psi_2(2) = 1$
$= \max [\, \underline{1/2 \cdot 5/6}, 1/2 \cdot 2/6 \,]$
$= 5/12$

$\delta_1(3) = \max [\, \delta_1(2) \cdot a_{11}, \delta_2(2) \cdot a_{21} \,]$ $\qquad \psi_1(3) = 2$
$= \max [\, 4/12 \cdot 1/6, \underline{5/12 \cdot 4/6} \,]$
$= 20/72$

$$\delta_2(3) = \max [\, \delta_1(2) \cdot a_{12}, \delta_2(2) \cdot a_{22} \,] \qquad \psi_2(3) = 1$$
$$= \max [\, \underline{4/12 \cdot 5/6}, 5/12 \cdot 2/6 \,]$$
$$= 20/72$$

$$\delta_1(4) = \max [\, \delta_1(3) \cdot a_{11}, \delta_2(3) \cdot a_{21} \,] \qquad \psi_1(4) = 2$$
$$= \max [\, 20/72 \cdot 1/6, \underline{20/72 \cdot 4/6} \,]$$
$$= 80/432$$

$$\delta_2(4) = \max [\, \delta_1(3) \cdot a_{12}, \delta_2(3) \cdot a_{22} \,] \qquad \psi_2(4) = 1$$
$$= \max [\, \underline{20/72 \cdot 5/6}, 20/72 \cdot 2/6 \,]$$
$$= 100/432$$

At the final time step we find that $\delta_2(4)$ has a higher probability than $\delta_1(4)$. This means that the optimal path must end in state S_2. In $\psi_2(4)$ we find the previous optimal state namely S_1, so we must continue our backtracking search in $\psi_1(3)$. In $\psi_1(3)$our trace goes to state S_2 and in $\psi_2(2)$ to state S_1. The optimal state sequence is therefore S_1, S_2, S_1, S_2 which corresponds to the observation sequence <u>train, car, train, car</u>; this observation sequence is most likely to occur. Note that only at the final time step we know where the optimal path ends, so in order to be able to trace back the optimal state sequence, we are obliged to store the pointer to the optimal preceding state for all states starting at t=2.
Before we proceed with the next paragraph we will give you another example of a real life process that can be modelled with a Markov chain, namely a 'weather forecast'. Suppose you consider only three types of weather: sunny, rainy and cloudy. What is the probability of weather type 'X' tomorrow if today weather type 'Y' occurs? In this restricted way the problem can be modelled in a Markov chain with three states (weather types). The transition probabilities can for instance be calculated with the weather statistics from the last 50 years.

2.3 Recognition with Markov models

Consider the Markov chain of figure 2a. Suppose you are given a red ball each time the system is in state S_1, a blue ball if it is in state S_2 and a yellow ball if it is in state S_3. The Markov model of figure 2b has a different initial state distribution and a different transition matrix, but also for this model state S_1 is associated with a red ball, state S_2 with a blue ball and state S_3 with a yellow ball.

Both systems are able to generate the following observation sequence:

$$O(1..6) = R\ B\ Y\ R\ Y\ R$$

We may ask ourselves which model is more likely to generate this sequence: model 2a or model 2b? This question can be simply answered by looking at the probability of this observation sequence, given the model. We will denote this probability as $P(O|\lambda)$; O stands for the observation sequence and λ for the Markov model. This probability for model 2a and model 2b is:

$$P(O|\lambda_1) = 0.2 \cdot 0.6 \cdot 0.8 \cdot 0.6 \cdot 0.2 \cdot 0.6$$
$$= 0.006912$$

$$P(O|\lambda_2) = 0.1 \cdot 0.1 \cdot 0.1 \cdot 0.4 \cdot 0.2 \cdot 0.4$$
$$= 0.000032$$

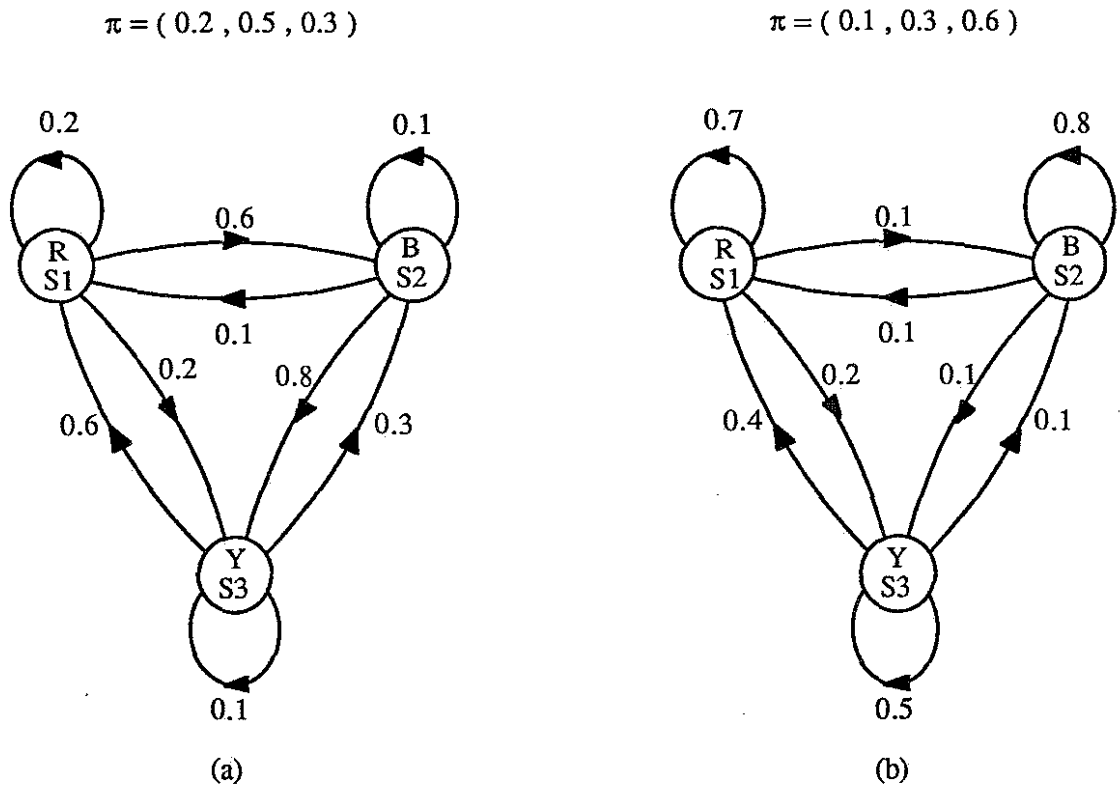

Figure 2. Two concurring Markov models used for recognition.

It will be clear that model 2a is more likely to generate this observation sequence. If on the other hand we look at another observation sequence

$$O(1..6) = R\,R\,Y\,Y\,Y\,B$$

and calculate the same probabilities for each model

$$P(O|\lambda_1) = 0.2 \cdot 0.2 \cdot 0.2 \cdot 0.1 \cdot 0.1 \cdot 0.3$$
$$= 0.000024$$

$$P(O|\lambda_2) = 0.1 \cdot 0.7 \cdot 0.2 \cdot 0.5 \cdot 0.5 \cdot 0.1$$
$$= 0.000350$$

we find that model 2b is more likely to generate the observation sequence. This is not surprising, because a closer look at the transition probabilities of the two models should convince you that model 2a has a tendency to constantly change colours in an observation sequence, whereas model 2b has a tendency to produce long strings of one colour. Although $P(O|\lambda)$ actually gives the probability of *generating* the observation sequence, we may interpret $P(O|\lambda_1) > P(O|\lambda_2)$ as a *recognition* of the observation sequence by model 2a and $P(O|\lambda_2) > P(O|\lambda_1)$ as a *recognition* of the observation sequence by model 2b. From this point of view model 2a is better at recognizing observation sequences in which the colour changes a lot and model 2b is better at recognizing sequences in which long strings of one colour occur.

It will be clear that the probabilities $P(O|\lambda)$ may become extremely small, especially if the observation sequence becomes longer or if the observation sequence is very unlikely for the model. To avoid underflow when a computer is used in the calculations it is better to calculate the *logarithm* of $P(O|\lambda)$ which can be done by *summing* the logarithmic transition probabilities since $\log(x \cdot y) = \log x + \log y$. A more elaborate discussion of this scaling problem is given in paragraph 3.6.

2.4 Training of Markov models

Now that we have seen how Markov models can recognize a sequence of observations, the next question is of course how do we obtain such Markov models? Imagine that there is some real life stochastic process that produces a sequence of red, blue and yellow balls. The sequence of colours is not at random, but it is constrained by some specific stochastic rules which we do not know. Furthermore we can obtain as many observation sequences as we like. How do we construct a Markov model from some of these observation sequences, that we will call the training set, so that we can recognize other observation sequences that originate from the same stochastic process? Suppose we have obtained the following training sequence:

$$O(1..30) = \text{Y R R Y Y Y Y B B B R R R B B Y Y Y B B B Y Y R R R B B R R}$$

Since there are three colours we choose a Markov model with three states, similar to the ones of figure 2. The transition probabilities can be estimated from the training sequence in the following steps:

1. Count the number of transitions $F_{ij}(T\text{-}1)$ from state S_i ($1 \le i \le N$) to state S_j ($1 \le j \le N$) for T-1 observations in the training set (for the last observation there is no transition).

2. Count the total number of transitions $F_i(T\text{-}1)$ from each state S_i ($1 \le i \le N$). This amounts to counting the number of times state S_i occurs (use also T-1 observations in this step).

3. Estimate a_{ij} as the number of transitions from step 1 divided by the number of transitions from step 2.

In a formula this can be expressed as:

$$\tilde{a}_{ij} = \frac{F_{ij}(T\text{-}1)}{F_i(T\text{-}1)} \qquad (5a)$$

In the following matrix the number of times each possible transition occurs is given (step 1):

	R	B	Y
R	6	2	1
B	2	6	2
Y	2	2	6

The row sums give the total number of transitions from each state (step 2). These row-sums are 9, 10 and 10 respectively . In order to get a proper transition matrix we must divide each cell of this matrix by its row sum (step 3):

$$A = \begin{pmatrix} 0.67 & 0.22 & 0.11 \\ 0.20 & 0.60 & 0.20 \\ 0.20 & 0.20 & 0.60 \end{pmatrix}$$

Since the observation sequence starts with a yellow ball the initial state distribution is estimated as

$$\hat{\pi} = (0, 0, 1)$$

In general the initial state distribution is estimated as

$$\hat{\pi}_i = F_i(1) \tag{5b}$$

It is obvious that a better estimation of the transition probabilities is obtained if the number of observations in the training set increases. Instead of training the Markov model with one long sequence of observations, we can also train it with a number of short sequences. More sequences will give the opportunity to make a better estimation of the initial state probabilities by counting the number of times the observation sequences start with a certain colour and dividing by the total number of sequences. There are two possible ways of averaging the a_{ij}'s of separate observation sequences; both satisfy the stochastic constraint that the a_{ij}'s summed over j ($1 \le j \le N$) are 1. The first possibility is:

$$\hat{a}_{ij} = \frac{\sum_{v=1}^{W} F_{ij}^{v}(T-1)}{\sum_{v=1}^{W} F_{i}^{v}(T-1)} \tag{6a}$$

where W is the number of observation sequences. $F_{ij}^{v}(T-1)$ denotes the total number of transitions from state S_i to state S_j in observation sequence v and $F_i^v(T-1)$ denotes the total number of times state S_i occurs in observation sequence v. The second possibility is:

$$\hat{a}_{ij} = \frac{1}{W} \sum_{v=1}^{W} \frac{F_{ij}^{v}(T-1)}{F_{i}^{v}(T-1)} \tag{6b}$$

Consider an observation sequence with 100 occurrences of state S_i and 10 transitions from state S_i to state S_j, and another observation sequence with 2 occurrences of state S_i and 1 transition from state S_i to state S_j. According to formula (6a) a_{ij} is calculated as $(10+1)/(100+2) = 11/102$ and according to formula (6b) as $1/2 \cdot (10/100 + 1/2) = 3/10$. Since the observation sequence with 100 occurrences probably gives a more reliable estimate of a_{ij}, formula (6a) that weighs frequencies is preferred (11/102 is closer to 10/100 than 3/10).

The formula for the initial state distribution becomes:

$$\pi_i = \frac{1}{W} \sum_{v=1}^{W} F_i^v(1) \qquad (6c)$$

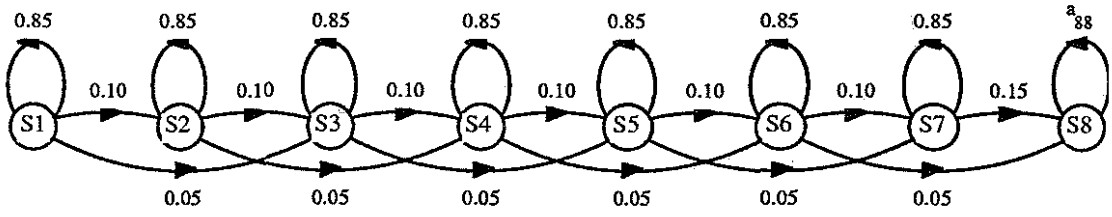

Figure 3. A left-to-right Markov model.

2.5 Left-to-right Markov models

Before we proceed with a discussion of hidden Markov models, there is one more topic that needs some attention, namely that of *left-to-right* Markov models. So far we have considered models in which it is possible to reach any state from any other state. These models are called *ergodic* models. The Markov model of figure 3 is a left-to-right model. This model has a distinct temporal structure in which a low state index always precedes a higher state index. Most of the transition matrix of a left-to-right model is thus filled with zeros. If we choose a_{88} to be 1.0, state S_8 is an *absorbing state*, because the system cannot get out of this state once it has been entered. For a further discussion of a_{88} see paragraph 4.2.2. Left-to-right Markov models are very useful to model time-varying signals such as speech.

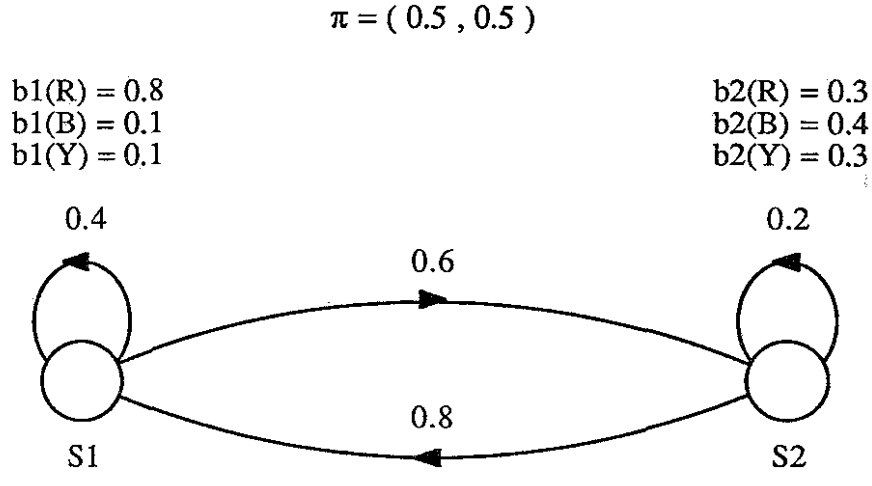

Figure 4. A simple hidden Markov model with two states and three observation symbols (colours).

3 HIDDEN MARKOV MODELS

3.1 Introduction to hidden Markov models

Suppose we want to model a stochastic process that generates 100 colours instead of 3 as in paragraph 2.3 (in chapter 4 we will actually use 64 different 'symbols' for the recognition of Dutch digits). With a 'normal' Markov model we would need 100 states, one for each colour. To obtain such a model we would have to train 10000 (100^2) transition probabilities, which would require an enormous amount of training data. The number of states can be drastically reduced with the use of *hidden Markov models* (HMM's), although at the cost of a greater arithmetic complexity as we will see.

Consider the Markov model of figure 4. This model has two states S_1 and S_2. However, the states are not associated with one coloured ball any more, but with an urn that is filled with an infinite number of balls. There are red balls, blue balls and yellow balls, each with a certain probability of occurrence and with a different probability distribution for each state (see figure 4). This *observation symbol probability distribution* can also conveniently be arranged in a N x M matrix, in which N denotes the number of states and M the number of observation symbols (colours in our example):

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & \dots & \dots & b_{1M} \\ b_{21} & b_{22} & b_{23} & \dots & \dots & b_{2M} \\ b_{31} & b_{32} & b_{33} & \dots & \dots & b_{3M} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ b_{N1} & b_{N2} & b_{N3} & \dots & \dots & b_{NM} \end{pmatrix}$$

called the observation matrix. An observation symbol probability is also commonly expressed as $b_j(k)$ or $b_j(O(t))$ which denotes the probability of observing symbol k ($1 \le k \le M$) or O(t) in state S_j. For the Markov model of figure 4

$$B = \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.3 & 0.4 & 0.3 \end{pmatrix}$$

An observation sequence is *generated* in the following way:

1. Choose one of the states (urns) according to an initial state distribution π.
2. Take a ball from the selected urn and record its colour.
3. Choose a new state (urn) according to the transition probability distribution.
4. Go back to 2, unless the required number of observations has been reached (t=T).

The observation sequence contains no 'visible' information about the state sequence, since each colour could have been generated by any state. In other words the state sequence is *hidden*. This type of model is therefore called a *hidden* Markov model.
Because each state of a hidden Markov model can accommodate all the observation symbols, we are in principle free to choose the number of states that we want to use for modelling a real life process (contrary to the Markov models of the previous chapter that

needed one state for each observation symbol). We may for instance model the stochastic process mentioned at the beginning of this paragraph with a 5-state hidden Markov model, which requires the training of 25 (5^2) transition probabilities and 500 (5 · 100) observation symbol probabilities, a total of 525 which is considerably smaller than the 10000 transition probabilities needed for the Markov models of the previous chapter. The problem of choosing the proper number of states for a hidden Markov model will be discussed in chapter 4.

3.2 Recognition with hidden Markov models

If we want to *recognize* observation sequences with hidden Markov models there are two questions of interest:

1. How do we calculate the probability of the observation sequence given the model $P(O|\lambda)$?

2. What is the most likely state sequence given the observation sequence and the model?

In principle the calculation of $P(O|\lambda)$ is sufficient for recognition. However, we can also use the probability of the most likely state sequence to recognize an observation sequence. Whereas $P(O|\lambda)$ is more accurate, the probability of the most likely path is easier to calculate. For speech recognition both types of probabilities are used.
To answer the first question we must calculate the probability of each possible path that accounts for the T observations O(1..T) and add the probabilities of all these paths. For this purpose we can use the forward procedure as described in paragraph 2.2 with a slight adjustment to incorporate the observation symbol probabilities. We now define $\alpha_j(t)$ as the joint event that O(1..t) are observed and the system stops in state S_j at time t. The recursion formula for $\alpha_j(t)$ becomes:

$$\alpha_j(t) = \sum_{i=1}^{N} \left(\alpha_i(t-1) \cdot a_{ij}\right) \cdot b_j(O(t)) \qquad (7)$$

in which $b_j(O(t))$ is the probability that O(t) (which represents one of the observation *symbols*) occurs in state S_j. The probability of the observation sequence given the model is calculated as

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_i(T) \qquad (8)$$

which is the sum of all α's at the final step T. To fix ideas we will illustrate the calculations for the hidden Markov model of figure 4 and the observation sequence

$$O(1..3) = R\ B\ Y$$

The α's are calculated as follows:

$$\begin{aligned}\alpha_1(1) &= \pi_1 \cdot b_1(R) \\ &= 0.5 \cdot 0.8 \\ &= 0.4\end{aligned}$$

$$\begin{aligned}\alpha_2(1) &= \pi_2 \cdot b_2(R) \\ &= 0.5 \cdot 0.3 \\ &= 0.15\end{aligned}$$

$$\begin{aligned}\alpha_1(2) &= [\, \alpha_1(1) \cdot a_{11} + \alpha_2(1) \cdot a_{21} \,] \cdot b_1(B) \\ &= [\, 0.4 \cdot 0.4 + 0.15 \cdot 0.8 \,] \cdot 0.1 \\ &= 0.028\end{aligned}$$

$$\begin{aligned}\alpha_2(2) &= [\, \alpha_1(1) \cdot a_{12} + \alpha_2(1) \cdot a_{22} \,] \cdot b_2(B) \\ &= [\, 0.4 \cdot 0.6 + 0.15 \cdot 0.2 \,] \cdot 0.1 \\ &= 0.108\end{aligned}$$

$$\begin{aligned}\alpha_1(3) &= [\, \alpha_1(2) \cdot a_{11} + \alpha_2(2) \cdot a_{21} \,] \cdot b_1(Y) \\ &= [\, 0.028 \cdot 0.4 + 0.108 \cdot 0.8 \,] \cdot 0.1 \\ &= 0.00976\end{aligned}$$

$$\begin{aligned}\alpha_2(3) &= [\, \alpha_1(2) \cdot a_{12} + \alpha_2(2) \cdot a_{22} \,] \cdot b_2(Y) \\ &= [\, 0.028 \cdot 0.6 + 0.108 \cdot 0.2 \,] \cdot 0.3 \\ &= 0.01152\end{aligned}$$

Therefore

$$\begin{aligned}P(O|\lambda) &= \alpha_1(3) + \alpha_2(3) \\ &= 0.02128\end{aligned}$$

is the probability of this observation sequence given the model of figure 4. Note that the sum over t ($1 \le t \le T$) for $\alpha_i(t)$ is no longer equal to 1 due to the second probabilistic layer of $b_j(k)$'s (compare paragraph 2.2).

To answer the second question we can use the Viterbi algorithm as described in paragraph 2.2 with the same adjustment as for the forward procedure to incorporate the observation symbol probabilities:

$$\delta_j(t) = \max_{1 \le i \le N} \left(\delta_i(t-1) \cdot a_{ij} \right) \cdot b_j(O(t)) \qquad (9a)$$

$$\psi_j(t) = \operatorname{argmax}_{1 \le i \le N} \left(\delta_i(t-1) \cdot a_{ij} \right) \qquad (9b)$$

The single best path with the highest probability for our example is calculated as follows:

$$\delta_1(1) = \pi_1 \cdot b_1(R) = 0.5 \cdot 0.8 = 0.4$$

$$\delta_2(1) = \pi_2 \cdot b_2(R) = 0.5 \cdot 0.3 = 0.15$$

$$\delta_1(2) = \max [\, \delta_1(1) \cdot a_{11}, \delta_2(1) \cdot a_{21} \,] \cdot b_1(B) \qquad \psi_1(2) = 1$$
$$= \max [\, \underline{0.4 \cdot 0.4}, 0.15 \cdot 0.8] \cdot 0.1 = 0.016$$

$$\delta_2(2) = \max [\, \delta_1(1) \cdot a_{12}, \delta_2(1) \cdot a_{22} \,] \cdot b_2(B) \qquad \psi_2(2) = 1$$
$$= \max [\, \underline{0.4 \cdot 0.6}, 0.15 \cdot 0.2 \,] \cdot 0.4 = 0.096$$

$$\delta_1(3) = \max [\, \delta_1(2) \cdot a_{11}, \delta_2(2) \cdot a_{21} \,] \cdot b_1(Y) \qquad \psi_1(3) = 2$$
$$= \max [\, 0.016 \cdot 0.4, \underline{0.096 \cdot 0.8} \,] \cdot 0.1 = 0.00768$$

$$\delta_2(3) = \max [\, \delta_1(2) \cdot a_{12}, \delta_2(2) \cdot a_{22} \,] \cdot b_2(Y) \qquad \psi_2(3) = 2$$
$$= \max [\, 0.016 \cdot 0.6, \underline{0.096 \cdot 0.2} \,] \cdot 0.3 = 0.00576$$

Since $\delta_1(3) > \delta_2(3)$ the optimal path ends in state S_1. According to $\psi_1(3)$ our trace goes back to state S_2 and in $\psi_2(2)$ to state S_1. The single best path with the highest probability given the observation sequence R B Y is therefore the state sequence S_1, S_2, S_1.

3.3 Training of hidden Markov models

How do we train the hidden Markov models? The simple procedure of estimating transition probabilities by counting frequencies from paragraph 2.4 cannot be used, because the observation sequence is no longer associated with only one state sequence. Furthermore the B-matrix has to be trained as well. However, if we replace frequencies of occurrence by probabilities, the principle of paragraph 2.4 is still valid. The following estimates are proposed for a_{ij}, $b_j(k)$ and π_i:

$$\tilde{a}_{ij} = \frac{\left(\begin{array}{c}\text{Probability of being in state } S_i \\ \text{and making a transition from state } S_i \text{ to state } S_j\end{array}\right)}{\text{Probability of being in state } S_i} \qquad (10a)$$

$$\tilde{b}_j(k) = \frac{\text{Probability of being in state } S_j \text{ and observing symbol } k}{\text{Probability of being in state } S_j} \qquad (10b)$$

$$\tilde{\pi}_i = \text{Probability of being in state } S_i \text{ at } t = 1 \qquad (10c)$$

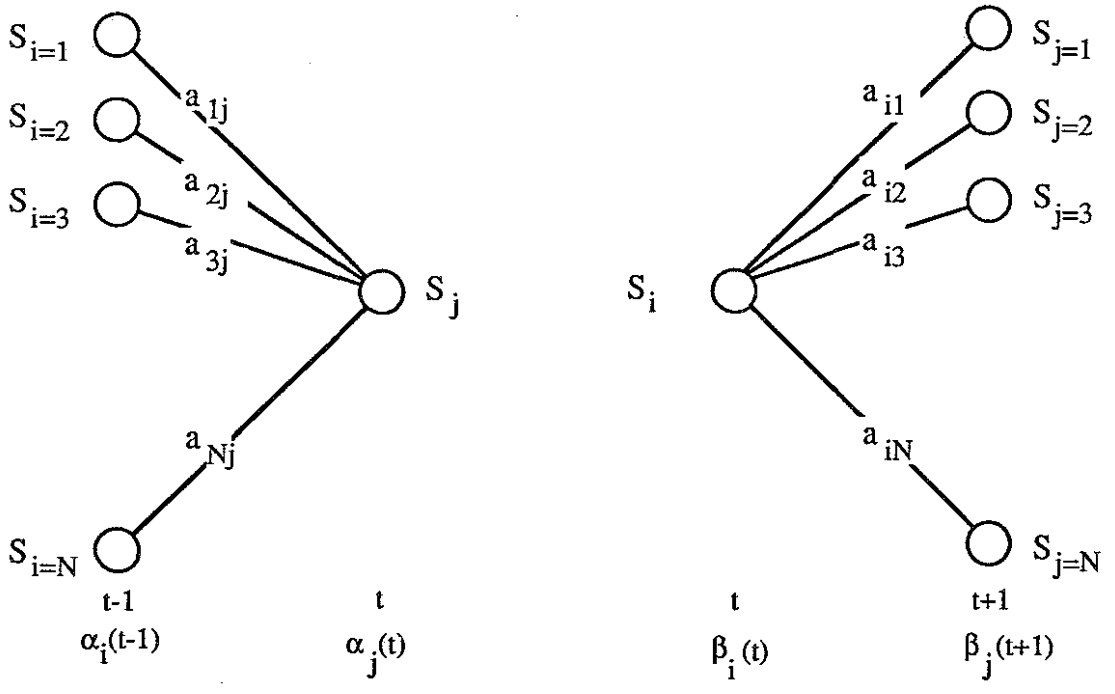

Figure 5. Illustration of one step of the forward procedure and one step of the backward procedure.

In order to calculate these probabilities we must introduce a new algorithm called the *backward procedure* which is the reverse of the forward procedure. Whereas $\alpha_j(t)$ is the probability of the joint event that O(1..t) are observed and the system *stops* in state S_j at time t, $\beta_i(t)$ is the probability of the joint event that O(t+1..T) are observed and the system *starts* in state S_i at time t (see figure 5):

$$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) \cdot a_{ij} \cdot b_j(O(t+1)) \qquad (11)$$

In a similar way as $\alpha_j(1)$ is initiated by the initial state distribution $\beta_i(T)$ is arbitrarily defined to be 1 for all i.
The probability of the joint event that O(1..T) are observed and the system *is* in state S_i at time t, which is mentioned in formula (10c) and in the denominator of formula (10a) and (10b), can therefore be defined as

$$\gamma_i(t) = \frac{\alpha_i(t) \cdot \beta_i(t)}{P(O|\lambda)} \qquad (12)$$

The normalization factor $P(O|\lambda)$, which can be calculated with the forward procedure as we have seen, makes $\gamma_i(t)$ a conditional probability so that the sum of all γ's at time t is 1. Next we define $\xi_{ij}(t)$ as the probability of the joint event that O(1..T) are observed and being in state S_i at time t and making a transition to state S_j at time t+1. This probability is mentioned in the numerator of formula (10a) and can be calculated as

$$\xi_{ij}(t) = \frac{\alpha_i(t) \cdot \beta_j(t+1) \cdot a_{ij} \cdot b_j(O(t+1))}{P(O|\lambda)} \tag{13}$$

So now we can estimate a_{ij} as

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{14a}$$

In the summation T is excluded because there is no transition from the final step (see paragraph 2.3). The probability $b_j(k)$, i.e. the probability that symbol k ($1 \leq k \leq M$) occurs when the system is in state S_j, can be estimated as

$$\bar{b}_j(k) = \frac{\sum_{\substack{t=1 \\ O(t)=k}}^{T} \gamma_j(t)}{\sum_{t=1}^{T} \gamma_j(t)} \tag{14b}$$

Since no transitions are involved the summation in this formula is from t=1 to t=T. A reasonable estimate of the initial state distribution is the probability of being in state S_i at t=1:

$$\bar{\pi}_i = \gamma_i(1) \tag{14c}$$

A closer look at the origin of the formulas (14a), (14b) and (14c) will reveal that the left-hand side of the equations also appears in the right-hand side. This means that we have to use an iterative procedure to improve the model parameters. We start with an initial guess for A, B and π and then we re-estimate the parameters until these values stop changing in two succeeding iteration steps (within certain limits). Formulas (14a), (14b) and (14c) which are used to train a hidden Markov model are called the *Baum-Welch re-estimation formulas*. The initial guesses for the matrices A, B and π can be random choices, normalized to satisfy the constraints

$$\sum_{i=1}^{N} a_{ij} = 1 \tag{15a}$$

$$\sum_{k=1}^{M} b_j(k) = 1 \tag{15b}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{15c}$$

An alternative initial guess could be a uniform distribution

$$a_{ij} = 1/N + \varepsilon \tag{16a}$$

$$b_j(k) = 1/M + \varepsilon \tag{16b}$$

$$\pi_i = 1/N + \varepsilon \tag{16c}$$

where ε is a small random distortion. The values a_{ij}, $b_j(k)$ and π_i must again be normalized according to the formulas (15a), (15b) and (15c).

3.4 Training with multiple observation sequences

The training of hidden Markov models with more than one observation sequence is quite straightforward. For each observation sequence separately the ξ's and γ's are calculated. When this is done for all observation sequences (which amounts to *one* iteration of the Baum-Welch procedure), the model parameters can be re-estimated with the following formulas (compare formulas (17a & c) with formulas (6a & c)):

$$\hat{a}_{ij} = \frac{\sum_{v=1}^{W} \sum_{t=1}^{T-1} \xi_{ij}^{v}(t)}{\sum_{v=1}^{W} \sum_{t=1}^{T-1} \gamma_i^{v}(t)} \tag{17a}$$

$$\hat{b}_j(k) = \frac{\sum_{v=1}^{W} \sum_{t=1}^{T} \gamma_j^{v}(t)}{\sum_{v=1}^{W} \sum_{t=1}^{T} \gamma_j^{v}(t)} \tag{17b}$$

$$\hat{\pi}_i = \frac{1}{W} \sum_{v=1}^{W} \gamma_i^{v}(1) \tag{17c}$$

where W denotes the number of words.

A scheme of the entire training procedure is shown in figure 6.

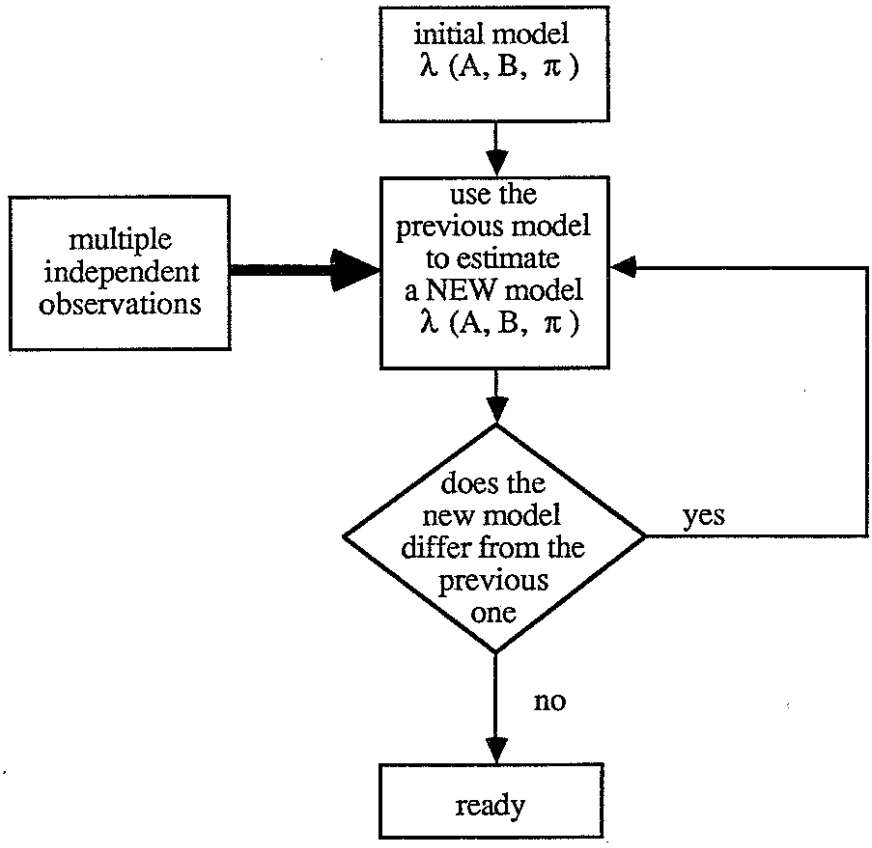

Figure 6. Scheme of training procedure.

3.5 Multiple symbol distributions

Before we discuss the topic of multiple symbols a remark has to be made about the nature of the symbols. In our example with coloured balls we assumed that there were only three distinct colours: Red, blue and yellow. Suppose the balls could have any colour that you like. This would request an infinite data set to train the Markov models. However, we might agree to assign each occurring colour to one of the colours red, blue or yellow (or any other *finite* set of colours). The assignment could be based on any reasonable criterion, e.g. the similarity of the most dominant frequency of the colours. In this function the colours red, blue and yellow are called the *codewords* and the total set of codewords is called the *codebook*. For the construction of codebooks a technique called *Vector Quantization* is often used, which will be described in paragraph 4.1.3. It is also possible to model *continuous* observation symbol distributions (e.g. Gaussian). However, the increased arithmetic complexity doesn't seem to outweigh the improvement in performance of Markov models using such distributions.
Let's assume that apart from the colour of the balls, also their weight is given. With Vector Quantization we have made three codewords for the weight-codebook of 100 gram, 200 gram and 300 gram. New balls, which have weights varying from a few grams to a few hundred grams, are assigned to the codeword that is closest in weight. How do we adapt the forward procedure to recognize observation sequences with multiple symbols and how do we adapt the training formulas?
The forward procedure has to account for the joint probability that observation symbol $O^1(t)$ (in our example a colour) and observation symbol $O^2(t)$ (a weight) occur. In general there may be Z different symbol distributions. The forward algorithm is therefore adapted in the following way:

$$\alpha_j(t) = \sum_{i=1}^{N} \left(\alpha_i(t-1)\cdot a_{ij}\right) \cdot b_j^1(O^1(t)) \cdot b_j^2(O^2(t)) \cdot \ldots \cdot b_j^Z(O^Z(t)) \qquad (18)$$

where $b_j^x(k)$ denotes the probability of observing symbol k of the x-th symbol distribution ($1 \le x \le Z$) in state S_j.
An alternative approach is to regard each $b_j^x(k)$ as a separate path, which means addition of all $b_j(k)$'s in formula (18). This gives a similar problem as the one discussed for formulas (6a) and (6b) in paragraph 2.4. Consider a hidden Markov model with two symbol distributions. In state S_j at time t we may have $b_j^1(k) = 0.9$ and $b_j^2(k) = 0.0$. This means that the occurrence $O^2(t) = k$ is impossible. This impossibility can only be expressed in the product of the probabilities $b_j^x(k)$; addition would unjustly lead to a high combined probability.
For the adaptation of the Baum-Welch re-estimation formula for a_{ij}, we must realize that in fact each new symbol distribution adds a new observation sequence to the training set. Therefore formula (17a) for multiple observations is also valid for multiple symbols:

$$\tilde{a}_{ij} = \frac{\sum_{x=1}^{Z} \sum_{t=1}^{T-1} \xi_{ij}^x(t)}{\sum_{x=1}^{Z} \sum_{t=1}^{T-1} \gamma_i^x(t)} \qquad (19a)$$

Since we want to determine Z different symbol distributions, each of these must be trained separately with the formula:

$$\tilde{b}_j^x(k) = \frac{\sum_{t=1}^{T} \gamma_j^x(t)}{\sum_{t=1}^{T} \gamma_j^x(t)} \qquad O^x(t)=k \qquad (19b)$$

where $b_j^x(k)$ denotes the probability of observing symbol k of the x-th symbol distribution ($1 \le x \le Z$) in state S_j.
For the initial probability distribution we use formula (17c) adapted for multiple symbols:

$$\tilde{\pi}_i = \frac{1}{Z} \sum_{x=1}^{Z} \gamma_i^x(1) \qquad (19c)$$

Note that in case of multiple observation sequences the formulas (19a,b,c) have to be extended with a summation over W observation sequences (and a multiplication with 1/W for formula (19c)).

3.6 Scaling of probabilities

It was already noted in paragraph 2.3 that the probabilities used in Markov models may become extremely small. It would therefore be preferable to use the logarithm of probabilities. This is easy to do if multiplications or divisions of probabilities are involved since

$$\log (x \cdot y) = \log x + \log y \tag{20a}$$

$$\log (x/y) = \log x - \log y \tag{20b}$$

So the transformation to logarithmic probabilities will give no problems for the formulas in which only multiplications and divisions of probabilities occur. This is not the case if additions of probabilities have to be calculated as for instance with the forward procedure or the backward procedure. However, a simple solution is to use scaling factors to keep the probabilities within the dynamic range of the computer and then later combine the scaling factors and the scaled probabilities to logarithmic probabilities. This technique will be demonstrated for the forward procedure. First each $\alpha_i(t)$ is divided by a scaling factor K_t before proceeding with the next time step $\alpha_i(t+1)$ and the scaling factor K_t is remembered. After all scaled $\alpha_i(t)$'s ($1 \leq t \leq T$) have been calculated the unscaled $\alpha_i(t)$'s are represented by

$$\text{unscaled } \alpha_i(t) = K_1 \cdot K_2 \cdot .. \cdot K_t \cdot \text{scaled } \alpha_i(t) \tag{21a}$$

For logarithmic values this amounts to

$$\log[\text{unscaled } \alpha_i(t)] = \log[K_1] + \log[K_2] + ... + \log[K_t] + \log[\text{scaled } \alpha_i(t)] \tag{21b}$$

The scaling factor we chose was the sum of all $\alpha_i(t)$'s for a fixed time t.

4 MARKOV MODELLING OF SPOKEN DUTCH DIGITS

4.1 Speech pre-processing

4.1.1 Filterbank

Before adapting HMM's to speech, we will outline how the pre-processing of the incoming speech signal is done (see first 4 blocks of figure 7). The recordings are done with a Sennheiser MD421N microphone. The continuous signal is low-pass filtered at 4.5 kHz (48 dB/octave), and then sampled at 16 kHz. Next a spectral analysis is performed using a FIR-based filterbank, resulting every 8 msec in energy estimates for 15 output channels. For details about this filterbank see Van Alphen et al. (1988). This filterbank has a constant bandwidth for the low-frequency region, and a 1/3 octave spacing for the high-frequency region (roughly Bark-scaled). To remove the harmonic structure in the lower filterbands, an f0-correction is applied using a heuristic method (Van Alphen, 1989).

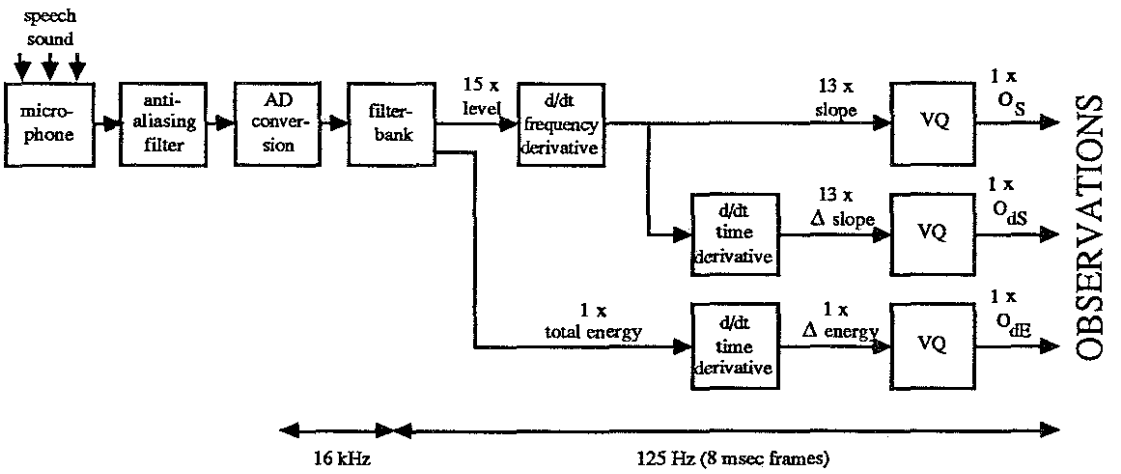

Figure 7. Pre-processing of speech signal.

4.1.2 Metrics

In recognition we do not use the direct Level metric L (spectrum consisting of 15 output energy levels), but the frequency derivative of the spectrum. We will call this variable the Slope metric S (it is a reduced form of Klatt's Weighted Spectral Slope Metric (Klatt, 1982)). This slope metric for filterband f at time t is defined as:

$$S_f(t) = L_{f+1}(t) - L_{f-1}(t) \qquad 2 \le f \le 14 \qquad (22)$$

where $L_f(t)$ is the energy level of filterband f at time t. In this definition an implicit smoothing of the spectrum is done. In this way we end up with a vector of 13 S-values for every time-frame. Although this slope metric will describe the stationary part of the speech signal rather well, it will do a bad job for transitions. For better modelling of these transitions, we defined two metrics similar to the ones used by Lee (Lee, 1989), namely ΔS (the time derivative of the slope metric) and ΔE (the time derivative of the total Energy). These are defined as

$$\Delta S_f(t) = S_f(t+2) - S_f(t-2) \qquad 3 \le t \le T-2 \qquad (23)$$

$$\Delta E(t) = E(t+2) - E(t-2) \qquad 3 \le t \le T-2 \qquad (24)$$

where T is the duration of the utterance, and E(t) the total Energy at time t.
With the 15 energy levels of the filterbank as base, we end up with the following metrics for modelling:

- S = Slope : frequency derivative of level metric 13 values
- ΔS = Δ Slope : time derivative of slope metric 13 values
- ΔE = Δ Energy : time derivative of total energy 1 value

Figure 7 shows that we started with 16000 data points per second, being the sample frequency of the speech signal. With the three defined metrics, we have 3375 data points per second: 13+13+1 values every 8 msec. (125 Hz). Although we reduced the data stream, it is still not suitable as input for our hidden Markov models. To achieve a

further data reduction, we will apply Vector Quantization (VQ). This VQ yields for each of the parameter vectors *one* output (observation) symbol, which has the same function as the colours red, blue and yellow from the urn model of paragraph 3.1.
For speech, this observation is a pointer to the closest vector from the Vector Quantization codebook. The way this codebook is obtained will be outlined in the next paragraph.

4.1.3 Vector Quantization

Vector Quantization is a technique that reduces a large number of n-dimensional data points to a small set of n-dimensional vectors called the codewords; the total set of codeword vectors is called the codebook. A widely used Vector Quantization, that is based on a very simple clustering algorithm, consists of the following sequence of steps:

1. Choose the number of codewords you want to use.

2. Start with an initial guess of the codewords.

3. Allocate each data point to the nearest codeword on the basis of a distance measure.

4. Compute new codewords as the centroids of each codeword cluster of data points.

5. Alternate step 3 and 4 until no data points change their cluster membership at step 3.

It can be shown that this iterative clustering process converges to a locally optimum quantizer (Anderberg, 1973). A variation on this method is to update the centroids of the losing and gaining clusters each time a data point is allocated to a new codeword cluster. However, the set of codewords constructed in this way will depend on the sequence in which the data points are processed. Since it is not clear how this will effect the clustering, we preferred to keep the codewords fixed for a full cycle through the data set.
For the initial guess of our codewords we applied a hierarchical clustering method to a small random subset of our data. The hierarchical clustering was the Ward method that merges at each stage two clusters on the basis of a variance criterion. The details of this procedure can be found in Anderberg (1973).
For each of the three codebooks (S, dS and dE) we wanted to make 64 codewords, a number that was used by Rabiner et al. (1983) in a similar digit recognition experiment. We started with 12500 data points (125 frames x 10 digits x 10 repetitions). A random set of 1000 data points from these data was used for the initial guess of the 64 codewords. The distance measure we used was the Euclidean distance.
In future research we plan to compare this Vector Quantization technique with other methods.

4.1.4 Summary of pre-processing

If the performed pre-processing (shown in figure 7) is considered as one block, the input is the (acoustic) speech signal, and the output is for every metric one observation (with a frame rate of 8 msec.). These observations are the observation symbols of the hidden Markov models of paragraph 3.1. If we consider all three metrics, there will be three observation sequences:

- $O^S(1..T)$ = pointer to the closest vector in the slope codebook
- $O^{\Delta S}(1..T)$ = pointer to the closest vector in the Δ slope codebook
- $O^{\Delta E}(1..T)$ = pointer to the closest vector in the Δ energy codebook

From now on, superscripts S, ΔS and ΔE will denote that the variable corresponds to one of the three metrics: slope, Δ slope and Δ energy respectively.

4.2 Hidden Markov models

4.2.1 Introduction

As an example of modelling speech with HMM's, we used the ten Dutch digits as vocabulary, and trained the HMM's for one single (male) speaker. Because we chose words as unit, the implementation of HMM's will be straightforward. We will outline this in the next paragraph, restricting ourselves to one metric (e.g. the slope metric). Once we have defined the HMM's, we will show in paragraph 4.2.3 how we can combine all metrics.

4.2.2 Word models

We created 10 hidden Markov models, one for every digit-word. The type of model we used was the left-to-right model with 8 states, only allowing self-state loops and transitions to a higher state-index (see figure 3). In this way the transition matrix A has 8 states (N=8). In general there are no clear criteria to choose a certain number of states for a model (Rabiner, 1983). We chose 8 states for the following reasons:

- the longest words contained 5 phonemes. Ideally we would expect that each state describes one phoneme.
- 2 states for the silences surrounding the word. A huge advantage of modelling silences is that in the training phase *no* segmentation has to be done.
- one end state with *no* transition probabilities. This means that a_{88} is equal to 0.0 in figure 3. In this way we avoid an absorbing state that tends to administer the duration inadequately.

The training words were spoken one by one (isolated words), preceded and followed by a silence. The duration of the spoken digits was always one second (including the silences), resulting in time t going from 1 up to 125 (=T).
The Vector Quantization yields one single observation for the slope metric every time frame. Because we have 64 different observations (paragraph 4.1.3), the observation matrix B has as dimensions 8 (=N) times 64 (=M).
To force the model to start in the left most state (state S_1), the initial state distribution vector π is zero for all values, except for state S_1, where it is 1.

4.2.3 Multiple symbol distributions

Now that we have described how to build a hidden Markov model for the slope metric, we must find a way to combine the three metrics we want to use. In fact, there are two possible ways of combining the metrics (Gupta, 1987). The first one, which one we will call *word level incorporation*, means that we simply build a separate HMM for each metric. The second one, which we will call *frame level incorporation*, means incorporation of the probabilities at frame level in a way described in paragraph 3.5 about multiple symbol distributions.

4.2.4 Recognition

Once we have the 10 (trained) models λ_d, one for each spoken digit d, recognition consists of running the forward algorithm. This means that we take the 10 HMM's one by one, and run the algorithm on the observation sequence O(1..T) (from one Vector Quantization). These 10 HMM's will give as a result the probabilities $p(O|\lambda_d)$, indicating how well the observation sequence can be explained by the model. Now we can pick the model that has the highest probability, and score the spoken word with the digit, that was the origin of that model.
When we want to incorporate the three different metrics slope, Δ slope and Δ energy, the following formula has to be used for word level incorporation:

$$p(O|\lambda_d) = p(O^S|\lambda_d^S) \cdot p(O^{\Delta S}|\lambda_d^{\Delta S}) \cdot p(O^{\Delta E}|\lambda_d^{\Delta E}) \qquad (25)$$

This amounts to multiplying the probabilities of the models for each metric.
Incorporation at frame level is described in paragraph 3.5. The forward algorithm of formula (18) becomes in case of the three metrics slope, Δ slope and Δ energy:

$$\alpha_j(t) = \sum_{i=1}^{N} \left(\alpha_i(t-1)\cdot a_{ij}\right) \cdot b_j^S(O^S(t)) \cdot b_j^{\Delta S}(O^{\Delta S}(t)) \cdot b_j^{\Delta E}(O^{\Delta E}(t)) \qquad (26)$$

Running this modified algorithm yields the total observation probability $p(O|\lambda_d)$.

4.2.5 Initialization and training

About the initialization of the hidden Markov parameters A, B and π, the following choices were made (see also paragraphs 3.3 & 4.2.2):

A according to the left-to-right model of figure 3.
B uniform as indicated in paragraph 3.3 (without random distortion). Because we have 64 different observation symbols, this results in a value 1/64 for every $b_j(k)$. One remark there has to be made about the "silent-states". Because they occur in every word twice, we decided to adjust by hand the initial estimates. This was done by looking at the observation symbols in silences, and picking the 5 most frequent observation symbols (k). These observation symbols $b_j(k)$ then got the value 1/10 (a normalization according to the formula (15b) has to be performed).
π the initial state distribution is zero for all values, except for state S_1, where it is 1. In the same way we modified the backward algorithm to force the model to end in the right most state S_N. This means that $\beta_j(T)$ is initialized to zero, except $\beta_N(T)$, which gets the value 1.

Now that we have an initial estimate, we can follow the scheme of figure 6 by (iteratively) training with multiple observations of a specific digit d. In our case we had a training set of 20 tokens for every digit .

In case of word level incorporation an HMM for each metric was trained separately. In case of frame level incorporation of the three metrics slope, Δ slope and Δ energy, the re-estimation formulas from paragraph 3.5 are applicable in a straightforward way (see formulas (19a, b, c)):

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1}\left(\xi_{ij}^{S}(t) + \xi_{ij}^{\Delta S}(t) + \xi_{ij}^{\Delta E}(t)\right)}{\sum_{t=1}^{T-1}\left(\gamma_{i}^{S}(t) + \gamma_{i}^{\Delta S}(t) + \gamma_{i}^{\Delta E}(t)\right)} \qquad (27a)$$

$$\bar{b}_{j}^{x}(k) = \frac{\sum_{t=1}^{T} \gamma_{j}^{x}(t)}{\sum_{t=1}^{T} \gamma_{j}^{x}(t)} \qquad (27b)$$

where superscript x denotes one of the three metrics S, ΔS and ΔE.

$$\bar{\pi}_{i} = \frac{\gamma_{i}^{S}(1) + \gamma_{i}^{\Delta S}(1) + \gamma_{i}^{\Delta E}(1)}{3} \qquad (27c)$$

4.3 Testing and results

Testing was done with 10 new realizations per digit, insuring independence of the training- and test-set. In table 1 the results of all performed tests are gathered. Whenever an HMM *not* corresponding to the digit spoken in a test word had the highest probability, this was counted as an error.
To stress the importance of the amount of training data for HMM's, we varied the number of training words used to train a specific HMM. Performance is very poor if we only use 1 or 2 training words, while 20 training words seem to be sufficient.
All different metrics (codebooks), and their possible incorporation, were used to create HMM's. Because we had 20 training words per digit, we had to choose which ones to use whenever we wanted to train with less than 20 words. In this case we ran the training a few times for different subsets of training words, and then selected the subset of words that performed worst. From the results it can be seen that the slope- and Δ slope metric separately already perform well, while the Δ energy metric, only describing the envelope of the energy performs quite poorly.

Table 1. Error rate for the recognition of 10 spoken Dutch digits in isolation, for different metrics and varying number of training words (test set size =100).

number of training words	metric / codebook				
	separate codebooks			multiple codebooks	
	slope	Δ slope	Δ energy	word level	frame level
1	28 %	61 %	89 %	26 %	21 %
2	8 %	30 %	82 %	9 %	7 %
10	3 %	12 %	64 %	1 %	0 %
20	1 %	5 %	21 %	0 %	0 %

5 CONCLUSION AND FUTURE RESEARCH

Hidden Markov models can successfully be applied to speech recognition. At least for the simple task of speaker dependent isolated digit recognition, error rates close to zero can be achieved. As a next step the HMM's of the 10 Dutch digits were connected in a parallel network, that was able to recognize these 10 digits in a continuous speech task. As these preliminary results seem very promising, further research will focus on continuous speech recognition based on HMM's of phonemes. These models will then be used to recognize speech in a system, that will be capable of automatic banking and of supplying information on the banking facilities.

6 REFERENCES

Van Alphen, P., Horst, B. & Pols, L.C.W. (1988). "A fast algorithm for a FIR based filterbank, designed for the acoustic front-end of a speech recognizer", Proc. Speech '88, 7th FASE Symp., Edinburgh, Book 2, 677-682.

Van Alphen, P. & Pols, L.C.W. (1989). "A real-time FIR-based filterbank, as the acoustic front-end of a speech recognizer", Proc. Eurospeech '89, Paris, 621-624.

Anderberg, M.R. (1973). Cluster analysis for applications, Academic Press, New York, 395 pages.

Gupta, V.N., Lennig, M. & Mermelstein, P. (1987). "Integration of acoustic information in a large vocabulary word recognizer", Proc. IEEE-ICASSP '87, Dallas, 697-700.

Klatt, D.H. (1982). "Prediction of perceived phonetic distance from critical-band spectra: A first step", Proc. IEEE-ICASSP '82, Paris, 1278-1281.

Lee, K.F. (1989). Automatic Speech Recognition: The Development of the SPHINX System, Kluwer Academic publishers, Boston, 207 pages.

Rabiner, L.R., Levinson, S.E. & Sondhi, M.M. (1983). "On the application of Vector Quantization and hidden Markov models to speaker-independent, isolated word recognition", The Bell System Technical Journal 62(4), 1075-1105.

7 RECOMMENDED LITERATURE

Gupta, V.N., Lennig, M. & Mermelstein, P. (1987). "Integration of acoustic information in a large vocabulary word recognizer", Proc. IEEE-ICASSP '87, Dallas, 697-700.

Lee, K.F. (1989). Automatic Speech Recognition: The Development of the SPHINX System, Kluwer Academic publishers, Boston, 207 pages.

Levinson, S.E. (1987). "Continuous speech recognition by means of acoustic/phonetic classification obtained from a hidden Markov model", Proc. IEEE-ICASSP '87, Dallas, 93-96.

Lloyd, E. (1980). "Processes in discrete time: Markov chains", in Ledermann, W. (editor), Handbook of Applicable Mathematics, Probability, Vol. II, John Wiley & Sons Ltd., Chichester, 373-401.

Lipschutz, S. (1966). "Markov chains", in Schaum's Outline Series: Theory and problems of finite mathematics, Schaum Publishing co., New York, 233-256.

Mari, J.F. & Roucos, S. (1985). "Speaker independent connected digit recognition using hidden Markov models", Proc. SPEECH TECH '85, New York, 127-132.

Rabiner, L.R., Levinson, S.E. & Sondhi, M.M. (1983). "On the application of Vector Quantization and hidden Markov models to speaker-independent, isolated word recognition", The Bell System Technical Journal 62(4), 1075-1105.

Rabiner, L.R. & Juang, B.H. (1986). "An introduction to hidden Markov models", IEEE ASSP Magazine 3(1), 4-16.

Rabiner, L.R. (1988). "Mathematical foundations of hidden Markov models", NATO ASI Series, Vol. F46, Recent Advances in Speech Understanding and Dialog Systems, Springer-Verlag, Berlin Heidelberg, 183-205.

Rabiner, L. R., Wilpon, J.G. & Soong, F.K. (1989). "High performance connected digit recognition using hidden Markov models", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-37(8), 1214-1225.