# Chapter 2

# Statistical Learning

In this chapter we will examine the application of a statistical learning technique to the acquisition of phonotactic models. The learning technique which we will use is called Hidden Markov Models. The chapter will start with an introduction to standard Markov models. After this we will examine the more elaborate Hidden Markov Models (HMMs). The experiments we will perform with HMMs have been divided in two groups. The first group of experiments consists of test experiments with a small data set. With these experiments we will try to find out what restrictions we have to impose on the HMMs and their training data in order to be able to make them acquire phonotactic knowledge. These experiments will be discussed in the third section. The fourth section will present the results of the HMMs that were applied to our main data set. The final section of the chapter will give some concluding remarks.

## 1 Markov models

Before discussing the Hidden Markov Models, we will present the standard Markov models. We start with a general description of these models. After this we will introduce two basic algorithms which are used in connection with Markov models: the forward procedure and the Viterbi algorithm.

### 1.1 General description of Markov models

A Markov model is a model consisting of states and weighted transitions. The task of a Markov model is recognizing or producing sequences. An example of a Markov model is shown in figure 2.1. It shows some popular target locations in a tourist walk

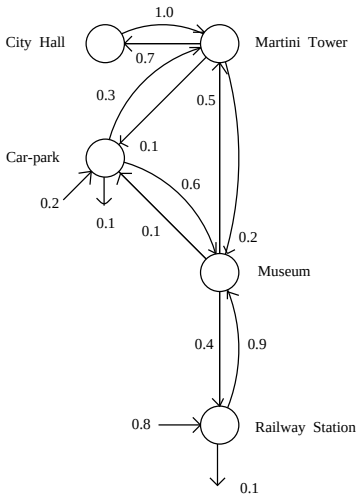


Figure 2.1: The City Walk Markov Model: A Markov model for a tourist walk through the city of Groningen. The states modeling locations are represented by filled circles and the transitions by arrows connecting them. The weights indicate the probability of moving from one state to another, e.g. when you are in the Museum the probability of moving to the Martini Tower is 0.5.

through the city of Groningen: the Museum (MU), the City Hall (CH) and the Martini Tower (MT). People start their walk from the Railway Station (RS) or from the Car park (CP).

These five locations are represented as states in the model. The weights of the links between the states indicate the probability that a visitor of a certain location will go to another location. 60% of the people present in the Car park at time *t* will be in the Museum at time *t+1* (0.6 link). All people in the City Hall will walk to the Martini Tower (1.0 link) and no people present in the Martini Tower will continue their walk by visiting the Railway Station next (no link). The starting locations of the walk are marked with links that start from outside the model. The probability that a walk starts in the Car park is 20% (0.2 link) and the probability that it starts in the Railway Station is 80% (0.8 link). A walk ends when the outward link from Car park (0.1) or Railway Station (0.1) is used.

The parameters of a Markov model can be arranged in matrices. One matrix, the A-matrix, contains the probabilities that concern the weights of internal links and an element $a_{ij}$ of this A-matrix indicates the probability that a transition between state *i* and state *j* will be made. The $\Pi$-matrix contains the probabilities of starting in states so $\pi_i$ indicates the probability that a Markov process starts in state *i*. Figure 2.2 shows

$$A = \begin{bmatrix} 0.00 & 0.90 & 0.00 & 0.00 & 0.00 \\ 0.40 & 0.00 & 0.10 & 0.50 & 0.00 \\ 0.00 & 0.60 & 0.00 & 0.30 & 0.00 \\ 0.00 & 0.20 & 0.10 & 0.00 & 0.70 \\ 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \end{bmatrix} \Pi = \begin{bmatrix} 0.80 \\ 0.00 \\ 0.20 \\ 0.00 \\ 0.00 \end{bmatrix}$$

Figure 2.2: Parameter definition of the City Walk Markov Model[1]. In the A-matrix a horizontal row $i$ contains the probabilities of leaving state $i$ and moving to a state $j$. The order of the states is <Railway Station, Museum, Car park, Martini Tower, City Hall> The $\Pi$-matrix contains the probabilities of starting in state $i$. Examples: The probability of moving from the Museum to the Railway Station is 0.40 (coordinate 2,1 in the A-matrix) and the probability of starting in the Car park is 0.20 (third number in $\Pi$-matrix.

the matrix representation for the City Walk Markov Model.

## 1.2 The forward procedure

Now suppose that the people that are engaged in a walk change location once an hour. What would be the probability that someone is in the Martini Tower after a walk of two hours? If we want to compute this probability we have to find out all paths from starting locations that reach the Martini Tower in two steps. Then we have to compute the probabilities of these paths and compute their sum. This will give us the probability we are looking for.

Inspection of figure 2.1 reveals that there are two paths possible from one of the starting locations which reach the Martini Tower in two steps: Car park $\rightarrow$ Museum $\rightarrow$ Martini Tower and Railway Station $\rightarrow$ Museum $\rightarrow$ Martini Tower. The probability of starting in Car park is 0.2, moving from Car park to Museum has probability 0.6 and moving from Museum to Martini Tower 0.5. Therefore the probability of the first path is $0.2 * 0.6 * 0.5 = 0.06$. In a similar fashion we can compute the probability of the second path: $0.8 * 0.9 * 0.5 = 0.36$. The probability that someone is in the Martini Tower after a walk of two hours is equal to the sum of these two probabilities: $0.06 + 0.36 = 0.42$.

The general procedure that is used for computing the probability that a Markov model will be in a state $s$ at a time $t$ is called the FORWARD PROCEDURE: $\alpha_s(t)$. The definition of this procedure is:

[1] In general the A-matrix and the $\Pi$-matrix satisfy the properties $\sum_j A[i,j] = 1$ and $\sum_i \Pi[i] = 1$. The first property is not satisfied by this A-matrix because we did not include the two model-leaving 0.1 arcs in the matrix. We will elaborate on this in section 3.1.

$$\begin{aligned}
\alpha_s(t) &= \textit{the probability of being in state } s \textit{ at time } t. \\
\alpha_s(0) &= \pi_s \qquad (2.1) \\
\alpha_s(t+1) &= \sum_k \alpha_{s_k}(t) * a_{s_k,s} \qquad (2.2)
\end{aligned}$$

Equation 2.1 defines that at time 0 the probability of being in a state *s* is equal to the probability of starting in state *s*. We can compute the probability of being in a state *s* at time point *t+1* if we know the probability of being a state at time *t*. First we have to multiply the probability of being in a state *k* with the weight of the link between state $s_k$ and state *s*. After this we have to compute the sum of all these products (equation 2.2). Weights of non-existent links are defined to have value *0*.

The forward procedure can be used for computing the probability that someone is in the Martini Tower after a walk of two hours. The procedure value we are looking for is $\alpha_{MT}(2)$. We start with computing all $\alpha_s(0)$. With these values we compute all $\alpha_s(1)$ values which in turn can be used for computing $\alpha_{MT}(2)$.

$$\begin{aligned}
\alpha_{RS}(0) &= \pi_{RS} = 0.8 \\
\alpha_{CP}(0) &= \pi_{CP} = 0.2
\end{aligned}$$

All other $\alpha_s(0)$ are equal to 0 because the probability of starting in these states ($\pi_s$) is 0. In the following equations we will assume that the states are ordered, that is: $s_1$=Railway Station (RS), $s_2$=Museum (MU), $s_3$=Car park (CP), $s_4$=Martini Tower (MT) and $s_5$=City Hall (CH). Now the next step is:

$$\begin{aligned}
\alpha_{RS}(1) &= \sum_k \alpha_{s_k}(0) * a_{s_k,RS} = 0 \\
\alpha_{MU}(1) &= \sum_k \alpha_{s_k}(0) * a_{s_k,MU} = \\
&= \alpha_{RS}(0) * a_{RS,MU} + \alpha_{CP}(0) * a_{CP,MU} = \\
&= 0.8 * 0.9 + 0.2 * 0.6 = 0.84 \\
\alpha_{CP}(1) &= \sum_k \alpha_{s_k}(0) * a_{s_k,CP} = 0 \\
\alpha_{MT}(1) &= \sum_k \alpha_{s_k}(0) * a_{s_k,MT} = \\
&= \alpha_{CP}(0) * a_{CP,MT} = \\
&= 0.2 * 0.3 = 0.06 \\
\alpha_{CH}(1) &= \sum_k \alpha_{s_k}(0) * a_{s_k,CH} = 0
\end{aligned}$$

$$\begin{aligned}\alpha_{leave}(1) &= \sum_k \alpha_{s_k}(0) * a_{s_k,leave} = \\ &= \alpha_{RS}(0) * a_{RS,leave} + \alpha_{CP}(0) * a_{CP,leave} = \\ &= 0.8 * 0.1 + 0.2 * 0.1 = 0.1\end{aligned}$$

There are no links from starting locations to the states Railway Station, Car park and City Hall. These locations cannot be reached in one step so their $\alpha_s(1)$ value is equal to *0*. The Martini Tower state can be reached from the Car park state in one step and the Museum state can be reached from both the Railway Station state and the Car park state. They have an $\alpha_s(1)$ value which is larger than *0*. The $leave$ state stands for leaving the city.

With the values of the forward procedure for time 1 we can now compute $\alpha_{MT}(2)$:

$$\begin{aligned}\alpha_{MT}(2) &= \sum_k \alpha_{s_k}(1) * a_{s_k,MT} = \\ &= \alpha_{MU}(1) * a_{MU,MT} = \\ &= 0.84 * 0.5 = 0.42.\end{aligned}$$

The Martini Tower can only be reached in two steps from one of the starting locations if the first step takes the tourist to the Museum. The probability $\alpha_{MT}(2)$ is equal to the one we have computed earlier.

By using the forward procedure we can speed up the computation of the probability of being in a state *s* at time *t*. For example, suppose that we have to perform such a computation for a ten-state Markov Model in which all states are connected with each other and in which each state can be a starting state. Suppose that we want to know a probability at time 10. If we compute this by summing the probabilities of all possible paths to the state we will get into trouble. We can start in ten different states (time 0) and at each new time point we can move to ten different states. This means that there are $10^{10}$ different paths to a state at time 10. Since every path from time 0 to time 10 requires 10 multiplications, we will need $10*10^{10}=10^{11}$. multiplications in order to compute the probability in this fashion.

With the forward procedure we can simplify this computation. At each time step we can compute each $a_s(t)$ with 10 multiplications (see equation 2.2). There are 10 $a_s(t)$, so at each time step we will need $10^2$ multiplications. For a probability at time 10 we will need 9 of those sets of multiplications and 10 additional multiplications for the final time slice. In total we need 910 multiplications with the forward procedure compared with the $10^{11}$ multiplications with the naive method.

## 1.3 The Viterbi algorithm

Now suppose that the city council wanted to put signs along a popular walk through Groningen. The budget of the council is limited, and they only have enough money

for a walk passing four locations. They want as many people as possible to see the signs so they want to know what four-step path starting from a starting location is the most likely one.

In total there are 48 4-step paths starting in one of the two initial positions. If we compute the probability of each of these paths, we will find out that the path Railway Station → Museum → Martini Tower → City Hall → Martini Tower is the most probable 4-step path (probability 0.252). We saw in the previous section that the number of paths can become quite large. In fact, the number of possible paths increases exponentially as path length increases. For longer paths computing the most likely one in this manner will take a lot of effort. Fortunately a good computational method exists for computing the most probable path: the VITERBI ALGORITHM. Its definition is:

$$\begin{aligned} \delta_s(t) &= \textit{the probability of the most likely path arriving in state } s \textit{ at time } t. \\ \delta_s(0) &= \pi_s \qquad (2.3) \\ \delta_s(t+1) &= max_k(\delta_{s_k}(t) * a_{s_k,s}) \qquad (2.4) \end{aligned}$$

The probability of the most likely path to a state at time 0 is equal to the probability of starting in that state (equation 2.3). Probabilities at other time points can be computed by using the probabilities of the previous time points (equation 2.4). Partial paths that cannot contribute to the most likely path are pruned by the *max* function. The Viterbi algorithm can be used to find out the most likely four-step path in the City Walk Markov Model. In this example it will have to compute 5*5=25 probabilities instead of the 48 probability computations that were necessary with the previous method. We start with computing $\delta_s(0)$:

$$\begin{aligned} \delta_{RS}(0) &= \pi_{RS} = 0.8 \\ \delta_{CP}(0) &= \pi_{CP} = 0.2 \end{aligned}$$

The other $\delta_s(0)$ are equal to 0 and have been left out. We continue by computing $\delta_s(1)$:

$$\begin{aligned} \delta_{MU}(1) &= max_k(\delta_{s_k}(0) * a_{s_k,MU}) = \\ &= max(\delta_{RS}(0) * a_{RS,MU}, \\ &\qquad \delta_{CP}(0) * a_{CP,MU}) = \\ &= max(0.8 * 0.9, 0.2 * 0.6) = 0.72 \\ \delta_{MT}(1) &= max_k(\delta_{s_k}(0) * a_{s_k,MT}) = \\ &= \delta_{CP}(0) * a_{CP,Martini\ Tower} = \\ &= 0.2 * 0.3 = 0.06 \end{aligned}$$

Again, the other $\delta_s(1)$ are equal to zero and have been left out. We will not compute $\delta_{leave}(t)$ because the probability of leaving the Markov model has no influence on the probabilities at later time points; in this model people that leave the city are not coming back. Here are the computations for $\delta_s(2)$

$$\begin{aligned}
\delta_{RS}(2) &= max_k(\delta_{s_k}(1) * a_{s_k,RS}) = \\
&= \delta_{MU}(1) * a_{MU,RS} = \\
&= 0.72 * 0.4 = 0.288 \\
\delta_{MU}(2) &= max_k(\delta_{s_k}(1) * a_{s_k,MU}) = \\
&= \delta_{MT}(1) * a_{MT,MU} = \\
&= 0.06 * 0.2 = 0.012 \\
\delta_{CP}(2) &= max_k(\delta_{s_k}(1) * a_{s_k,CP}) = \\
&= max(\delta_{MU}(1) * a_{MU,CP}, \\
&\qquad \delta_{MT}(1) * a_{MT,CP}) = \\
&= max(0.72 * 0.1, 0.06 * 0.1) = 0.072 \\
\delta_{MT}(2) &= max_k(\delta_{s_k}(1) * a_{s_k,MT}) = \\
&= \delta_{MU}(1) * a_{MU,MT} = \\
&= 0.72 * 0.5 = 0.36 \\
\delta_{CH}(2) &= max_k(\delta_{s_k}(1) * a_{s_k,CH}) = \\
&= \delta_{MT}(1) * a_{MT,CH} \\
&= 0.06 * 0.7 = 0.042
\end{aligned}$$

The computations of $\delta_s(3)$ and $\delta_s(4)$ can be performed in a similar fashion. We will not list the complete computations here but confine with the results:

$$\begin{aligned}
\delta_{RS}(3) &= 0.0048 \\
\delta_{MU}(3) &= 0.2592 \\
\delta_{CP}(3) &= 0.036 \\
\delta_{MT}(3) &= 0.042 \\
\delta_{CH}(3) &= 0.252 \\
\delta_{RS}(4) &= 0.10368 \\
\delta_{MU}(4) &= 0.0216 \\
\delta_{CP}(4) &= 0.02592 \\
\delta_{MT}(4) &= 0.252 \\
\delta_{CH}(4) &= 0.0294
\end{aligned}$$

| Location | Likes (L) | Dislikes (D) |
|---|---|---|
| Railway Station | 0.7 | 0.3 |
| Museum | 0.5 | 0.5 |
| Car park | 0.1 | 0.9 |
| Martini Tower | 0.9 | 0.1 |
| City Hall | 0.8 | 0.2 |

Figure 2.3: The probabilities of liking locations in the City Walk through Groningen. Probabilities are only dependent on the currently visisted object. We assume that previously visited locations do not influence the opinion of the tourists. Example: the probability that someone that visited the Martini Tower disliked it is 0.1 and the probability that the person liked it is 0.9.

As we can see $\delta_{MT}(4)$ has the largest value (0.25200) so the path starting at the Railway Station and ending at the Martini Tower is the most likely path containing 4 steps. This $\delta$-value only gives us the final point of this path. If we want to know the other locations present in the path, we will have to trace back the equations: $\delta_{MT}(4)$ most probable ancestor was $\delta_{CH}(3)$, its most probable ancestor was $\delta_{MT}(2)$ which on its turn was preceded by $\delta_{MU}(1)$ and $\delta_{RS}(0)$. So the most probable four-step path is Railway Station $\rightarrow$ Museum $\rightarrow$ Martini Tower $\rightarrow$ City Hall $\rightarrow$ Martini Tower and its probability is 0.252.

# 2 Hidden Markov Models

In the previous section we have examined Markov models. In this section we will present an extended version of these models: Hidden Markov Models. We will give a general description of these models and present the adapted versions of the forward procedure and the Viterbi algorithm which are used in Hidden Markov Models. After this we will describe how they can learn and how they can be used in practice.

## 2.1 General description of Hidden Markov Models

A researcher of the University of Groningen dedicates himself to finding out if people that take part in a city walk through Groningen like the locations they visit. To find the answer to this question he makes people fill in forms in which they are asked if they like the locations they have visited. The results of these forms are summerized in the table in figure 2.3.

The probability that someone likes the Railway Station (L) is 70% while the probability that the person does not like it (D) is 30%. All other probabilities are listed in the table. Now every walk through Groningen can be represented by a sequence con-

$$B = \begin{bmatrix} 0.70 & 0.30 \\ 0.50 & 0.50 \\ 0.10 & 0.90 \\ 0.90 & 0.10 \\ 0.80 & 0.20 \end{bmatrix}$$

Figure 2.4: An Extension of the parameter definition of the City Walk Markov Model: the B-matrix. In this matrix an entry $b_{s_i,t_j}$ represents the probability that in state $s_i$ token $t_j$ can be produced. Example: the probability of producting L (like, $t_1$) in the Car park state ($s_3$) is 0.10 and the probability of producing D (dislike, $t_2$) in the same state is 0.90. The A-matrix and the $\Pi$-matrix remain unchanged.

taining the tourist's opinion about the different locations. For example, if the previous most-probable walk Railway Station $\rightarrow$ Museum $\rightarrow$ Martini Tower $\rightarrow$ City Hall $\rightarrow$ Martini Tower was made by someone who likes the Martini Tower and the Museum but does not like the Railway Station, the Car park and the City Hall, we would get the sequence DLLDL.

The DL-sequences are an interesting by-product of the City Walk model. They are not in a unique correspondence with state sequences. For example a walk consisting of Car park $\rightarrow$ Martini Tower $\rightarrow$ Museum $\rightarrow$ Railway Station $\rightarrow$ Museum taken by the same tourist can also be represented with the sequence DLLDL. Furthermore, the same walk of a tourist that likes all locations can be represented by LLLLL. So different walks can be represented with the same DL-sequences and the same walk can generate different DL-sequences when made by different tourists.

It is possible to extend Markov Models to make them simulate this behavior. In order to do that we define that in a state of the model different tokens can be produced. The probabilities that tokens are produced will be stored in a new parameter matrix of the model: the B-matrix (figure 2.4). In this matrix an entry $b_{s_i,t_j}$ represents the probability that in state $s_i$ token $t_j$ can be produced. The other model parameters incorporated in the A-matrix and the $\Pi$-matrix remain the same as in figure 2.2.

Now we have obtained a HIDDEN MARKOV MODEL (HMM). The model is called *hidden* because from the token sequences generated by the model it is in general impossible to find out which states were passed through while generating the sequence. In the specific example of the City Walk Markov Model different state sequences could lead to the same DL-sequence. Therefore it was impossible to find out the state sequence used if we only know a token sequence.

## 2.2 The extended forward procedure

We have presented the forward procedure and the Viterbi algorithm for Markov models by asking two questions. We will do the same for the related functions for HMMs.

Our first Markov model question can be rephrased to: what is the probability of being at the Martini Tower after a walk of two hours in which the tourist did not like the starting location but in which he did like the second and the third location? In Hidden Markov Model terms, what is the probability of being at state Martini Tower after two steps while having produced token sequence DLL? We cannot use the forward procedure for Markov Models for computing this probability because it does not take into account the token production probabilities. A function that uses these probabilities is the EXTENDED FORWARD PROCEDURE:

$$\begin{aligned}
\alpha_s(t, x_0..x_t) &= \textit{the probability of being in state } s \textit{ at time } t \\
&\quad \textit{after producing sequence } x_0..x_t \\
\alpha_s(0, x_0) &= \pi_s * b_{s,x_0} \qquad (2.5) \\
\alpha_s(t+1, x_0..x_{t+1}) &= \sum_k \alpha_{s_k}(t, x_0..x_t) * a_{s_k,s} * b_{s,x_{t+1}} \qquad (2.6)
\end{aligned}$$

The probability of producing a token in a state at time $t = 0$ is equal to the probability of starting in that state multiplied with the probability of producing the token in the state (equation 2.5). The probability of producing a token $x_{t+1}$ in state s at time $t+1$ is equal to the sum of all values of the forward procedure for time $t$ and state $s_k$ multiplied with the probability of moving from state $s_k$ to $s$ and the probability of producing $x_{t+1}$ in $s$ (equation 2.6). Note that the extended forward procedure uses the probabilities of all tokens up to the current one. So the probability computed by this function is not only dependent on the current token but also on all previous tokens in the sequence.

We can use the extended forward procedure for answering the question we mentioned at the start of this section:

$$\begin{aligned}
\alpha_{RS}(0, D) &= \pi_{RS} * b_{RS,D} = \\
&= 0.8 * 0.3 = 0.24 \\
\alpha_{CP}(0, D) &= \pi_{CP} * b_{CP,D} = \\
&= 0.2 * 0.9 = 0.18 \\
\alpha_{MU}(1, DL) &= \sum_k \alpha_{s_k}(0, D) * a_{s_k,MU} * b_{MU,L} = \\
&= \alpha_{RS}(0, D) * a_{RS,MU} * 0.5 + \alpha_{CP}(0, D) * a_{CP,MU} * 0.5 = \\
&= 0.24 * 0.9 * 0.5 + 0.18 * 0.6 * 0.5 = 0.162 \\
\alpha_{MT}(1, DL) &= \sum_k \alpha_{s_k}(0, D) * a_{s_k,MT} * b_{MT,L} = \\
&= \alpha_{CP}(0, D) * a_{CP,MT} * 0.9 = \\
&= 0.18 * 0.3 * 0.9 = 0.0486
\end{aligned}$$

All $\alpha_s(0, x_0)$ and $\alpha_s(1, x_0..x_1)$ which are equal to 0 have been left out. Now we can compute the probability we are looking for:

$$\begin{aligned}\alpha_{MT}(2, DLL) &= \sum_k \alpha_{s_k}(1, DL) * a_{s_k,MT} * b_{MT,L} = \\ &= \alpha_{MU}(1, DL) * a_{MU,MT} * 0.9 = \\ &= 0.162 * 0.5 * 0.9 = 0.0729\end{aligned}$$

The probability which is the result of this computation is smaller than the $\alpha_{MT}(2)$ which was computed for the Markov Models (0.42) because of the multiplications with the token production probabilities ($b_{s_i,t_j} < 1$).

## 2.3 The extended Viterbi algorithm

We can also rephrase our second Markov Model question for Hidden Markov Models: what is the most probable two step walk in which the tourist liked all locations? Or in Hidden Markov Models terms what is the most probable path corresponding with the sequence LLL? We cannot use the Viterbi algorithm for Markov Models because it does not include token production probabilities. We have to adapt this function to obtain the EXTENDED VITERBI ALGORITHM:

$$\begin{aligned}\delta_s(t, x_0..x_t) &= \textit{the probability of the most likely path ending in state } s \textit{ and producing token sequence } x_0..x_t \\ \delta_s(0, x_0) &= \pi_s * b_{s,x_0} \quad (2.7) \\ \delta_s(t+1, x_0..x_{t+1}) &= max_k(\delta_{s_k}(t, x_0..x_t) * a_{s_k,s} * b_{s,x_{t+1}}) \quad (2.8)\end{aligned}$$

The probability of the most likely path that ends in state $s$ and produces a sequence containing one token $x_0$ is equal to the probability of starting in $s$ multiplied with the probability of producing $x_0$ in $s$ (equation 2.7). The probability of the most likely path ending in $s$ for a longer sequence is the maximal value that can be obtained by multiplying the probability of the most likely path for the prefix of the sequence with the probability of moving from the final state $s_k$ of the prefix to $s$ and the probability of producing the current token in $s$. The computation this extended Viterbi algorithm performs is sequence-specific just as the computation of the extended forward procedure.

This extended Viterbi algorithm can be used for finding out the most probable state path corresponding with the sequence LLL. First we compute the values of the $\delta$-function for time $t = 0$:

$$
\begin{aligned}
\delta_{RS}(0,L) &= \pi_{RS} * b_{RS,L} = \\
&= 0.8 * 0.7 = 0.56 \\
\delta_{CP}(0,L) &= \pi_{CP} * b_{CP,L} = \\
&= 0.2 * 0.1 = 0.02
\end{aligned}
$$

We can use values for computing the values of the functions for $t = 1$:

$$
\begin{aligned}
\delta_{MU}(1,LL) &= max_k(\delta_{s_k}(0,L) * a_{s_k,MU} * b_{MU,L}) = \\
&= max(\delta_{RS}(0,L) * a_{RS,MU} * 0.5, \delta_{CP}(0,L) * a_{CP,MU} * 0.5) = \\
&= max(0.56 * 0.9 * 0.5, 0.02 * 0.6 * 0.5) = 0.252 \\
\delta_{MT}(1,LL) &= max_k(\delta_{s_k}(0,L) * a_{s_k,MT} * b_{MT,L}) = \\
&= \delta_{CP}(0,L) * a_{CP,MT} * 0.9 = \\
&= 0.02 * 0.3 * 0.9 = 0.0054
\end{aligned}
$$

The $\delta_s(0,L)$ and $\delta_s(1,LL)$ that are equal to zero and have been left out. With these results we can compute all $\delta_s(2,LLL)$:

$$
\begin{aligned}
\delta_{RS}(2,LLL) &= max_k(\delta_{s_k}(1,LL) * a_{s_k,RS} * b_{RS,L}) = \\
&= \delta_{MU}(1,LL) * a_{MU,RS} * 0.7 = \\
&= 0.252 * 0.4 * 0.7 = 0.07056 \\
\delta_{MU}(2,LLL) &= max_k(\delta_{s_k}(1,LL) * a_{s_k,MU} * b_{MU,L}) = \\
&= \delta_{MT}(1,LL) * a_{MT,MU} * 0.5 = \\
&= 0.0054 * 0.2 * 0.5 = 0.00054 \\
\delta_{CP}(2,LLL) &= max_k(\delta_{s_k}(1,LL) * a_{s_k,CP}) * b_{CP,L} = \\
&= max(\delta_{MU}(1,LL) * a_{MU,CP} * 0.1, \\
&\qquad \delta_{MT}(1,LL) * a_{MT,CP} * 0.1) = \\
&= max(0.252 * 0.1 * 0.1, 0.0054 * 0.1 * 0.1) = 0.00252 \\
\delta_{MT}(2,LLL) &= max_k(\delta_{s_k}(1,LL) * a_{s_k,MT} * b_{MT,L}) = \\
&= \delta_{MU}(1,LL) * a_{MU,MT} * 0.9 = \\
&= 0.252 * 0.5 * 0.9 = 0.1134 \\
\delta_{CH}(2,LLL) &= max_k(\delta_{s_k}(1,LL) * a_{s_k,CH} * b_{CH,L}) = \\
&= \delta_{MT}(1,LL) * a_{MT,CH} * 0.8 = \\
&= 0.0054 * 0.7 * 0.8 = 0.003024
\end{aligned}
$$

From these computation we can conclude that the most probable path that produces $LLL$ ends in the Martini Tower. We can find out the previous locations by checking what state was used in the maximal part of the $\delta_{MT}(2, LLL)$ computation and this turns out to be the Museum ($\delta_{MU}(1, LL)$). The location before that can be found by checking what state was used in the maximal part of the $\delta_{MU}(1, LL)$ computation. This turns out to be the Railway Station ($\delta_{RS}(0, L)$) so the most probable path producing LLL is Railway Station $\rightarrow$ Museum $\rightarrow$ Martini Tower.

## 2.4 Learning in a Hidden Markov Model

Now suppose that in some distant country a group of engineers decides to rebuild the main tourist attractions of Groningen. The engineers also want to enable the visitors of New Groningen to experience the famous City Walk through Groningen. Unfortunately, the engineers do not know what the main buildings look like and which buildings were connected with each other. The only feature about the City Walk they were able to collect is a list of DL-sequences that were produced by participants in the City Walk through Groningen. The engineers decide to set up some wooden barracks with roads connecting them to each other and make tourists walk through this village. The tourists all take with them a form in which they mention what locations they visited and whether they liked the location. If the tourists thus produce a DL-sequence that is in the list the engineers are trying to reproduce, the engineers will do nothing. If, however, the DL-sequence is not in the list the engineers start improving or damaging the buildings and the roads. This process continues until the tourists only produce sequences that are in the list. The engineers have then succeeded in reproducing the City Walk through Groningen and they have succeeded in reconstructing the underlying Hidden Markov Model as far as the production of DL-sequences is concerned.

The problem of finding a Hidden Markov Model which produces a specific set of token sequences is a common task. Our goal is to obtain a Hidden Markov Model that produces sequences of characters. The model should assign high probabilities to sequences that are words in some language and low probabilities to sequences that cannot appear as words in the language. Note that we are not aiming at reproducing the exact underlying model for the language. We will try to find a model that *behaves* like the underlying language model. Like the engineers of New Groningen we only know the token sequences produced by the model we are trying to rebuild. We will be satisfied if we succeed in creating a model that is able to reproduce our data.

The problem is that there is no direct method for computing the parameters (the matrices A, $\Pi$ and B) of a Hidden Markov Model that is able to produce a specific set of sequences with a large probability. Fortunately, there are methods for estimating the values of the parameters of such a Hidden Markov Model. The most well-known method for estimating the parameters of a Hidden Markov Model from a set of sequences is called the BAUM-WELCH ALGORITHM or the forward-backward algorithm which has been described in (Rabiner et al. 1986) and (Van Alphen 1992) among others. This algorithm consists of three steps:

1. Initialize the Hidden Markov Model with random parameter values.
2. Make the Hidden Markov Model compute the probability of every sequence in the set. During this computation we count how often transitions between states are used and how often tokens are being produced in each state. We use the resulting numbers for computing a new set of parameters.
3. We use the new parameter values for reinitializing the Hidden Markov Model. The new Hidden Markov Model will assign a higher probability to the set of training strings. Now we repeat step 2 and 3 until the behavior of the Hidden Markov Model stabilizes.

There are different methods for deciding when a HMM has become stable. We will discuss these in a later section. It is possible to prove that the Baum-Welch algorithm terminates so we can be sure that the algorithm will always be able to produce a stable HMM in a finite amount of time. The proof is complex and we will not list it here. Interested readers are referred to section 5.4 of (Huang et al. 1990)

The most complex step of the Baum-Welch algorithm is step 2. We will formalize this step by using the extended forward procedure (see equations 2.5 and 2.6) and three other algorithms we will introduce in this section. Our goal is to find new values for the HMM parameters $a_{ij}$, $b_{s_i,m}$ and $\pi_{s_i}$. The definitions of these parameters are:

$$a_{s_i,s_j} = \frac{probability\ of\ making\ a\ transition\ from\ s_i\ to\ s_j}{probability\ of\ being\ in\ state\ s_i} \tag{2.9}$$

$$b_{s_i,m} = \frac{probability\ of\ being\ in\ s_i\ while\ producing\ token\ m}{probability\ of\ being\ in\ state\ s_i} \tag{2.10}$$

$$\pi_{s_i} = probability\ of\ being\ in\ state\ s_i\ at\ time\ 0 \tag{2.11}$$

In equations 2.9 and 2.10 it is necessary to divide the numerator probabilities by the probability of being in state $s_i$ to make sure that for each $s_i$ all $a_{s_i,s_j}$ add up to 1 and all $b_{s_i,m}$ add up to 1. In order to be able to compute new values for the A-matrix, we should be able to compute the probability that in a production of a sequence a specific transition between two states will be made. We can view the production of a sequence as consisting of three steps: the production of the current token, the production of the prefix of this token and the production of the suffix of the token.[2] We have an algorithm that models the production of the prefix of a token: the forward procedure. The first algorithm we will introduce here is the BACKWARD ALGORITHM: an algorithm that models the production of the suffix of a token ($e$ is the empty sequence and $T$ is the time at which the final element of the sequence is produced):

[2] In this chapter we do not use prefix and suffix as the linguistic terms. For us the prefix of a sequence is the subsequence from the start to the current token (non-inclusive) and the suffix a sequence is the subsequence from the current token (non-inclusive) to the end of the sequence.

$$\begin{aligned}
\beta_{s_i}(t, x_{t+1}..x_T) &= \textit{probability that a sequence } x_{t+1}..x_T \textit{ is produced} \\
&\quad \textit{while the state at time } t \textit{ is } s_i. \\
\beta_{s_i}(T, e) &= 1 \qquad (2.12) \\
\beta_{s_i}(t, x_{t+1}..x_T) &= \sum_j \beta_{s_j}(t+1, x_{t+2}..x_T) * a_{s_i,s_j} * b_{s_j,x_{t+1}} \qquad (2.13)
\end{aligned}$$

The backward algorithm is the counterpart of the extended forward algorithm. It computes the probability that a suffix of a sequence is produced while starting in a specific state ($s_i$). The probability that the empty string is produced in state $s_i$ at time $T$ is defined to be equal to one (equation 2.12). If we know the probabilities of a sequence which starts at time $t+1$ in any of the states $s_j$ then we can compute the probability of the same sequence preceded by some token at time $t$ and state $s_i$ by multiplying the known value with the probability of moving from $s_i$ to $s_j$ and the probability of producing $x_{t+1}$ in $s_j$ and adding all these products together. Note that in this definition the production of a token is imagined as happening directly after the transition to the state that produces the token. That is why equation 2.13 contains $b_{s_j,x_{t+1}}$ and not $b_{s_i,x_t}$ and why in 2.12 no token production has been taken into account.

We can combine the forward and the backward algorithm for computing the probability that the HMM assigns to a sequence:

$$\begin{aligned}
P_h(x_0..x_T) &= \textit{probability assigned to sequence } x_0..x_T \textit{ by an } HMM \\
P_h(x_0..x_T) &= \sum_i \alpha_{s_i}(t, x_0..x_t) * \beta_{s_i}(t, x_{t+1}..x_T) \qquad (2.14)
\end{aligned}$$

The term $\alpha_{s_i}(t, x_0..x_t) * \beta_{s_i}(t, x_{t+1}..x_T)$ computes the probability of being in state $s_i$ at time $t$ while producing sequence $x_0..x_T$. Equation 2.14 computes the sum for all $i$ which gives us the probability of producing sequence $x_0..x_T$ while being in an arbitrary state $s_i$ at time $t$. This is equal to the probability of producing the sequence $x_0..x_T$. We can use $P_h(x_0..x_T)$ for computing the probability of being in state $s_i$ at time $t$:

$$\begin{aligned}
\gamma_{s_i}(t) &= \textit{the probability of being in state } s_i \textit{ at time } t \\
\gamma_{s_i}(t) &= \frac{\alpha_{s_i}(t, x_0..x_t) * \beta_{s_i}(t, x_{t+1}..x_T)}{P_h(x_0..x_T)} \qquad (2.15)
\end{aligned}$$

We need to divide $\alpha_{s_i}(t, x_0..x_t) * \beta_{s_i}(t, x_{t+1}..x_T)$ by the probability of producing sequence $x_0..x_T$ in order to make sure that for each $t$ the $\gamma_{s_i}(t)$ probabilities sum up to 1. With this $\gamma_{s_i}(t)$ and the equations 2.10 and 2.11 we are now able to compute new values for $\pi_{s_i}$ and $b_{s_i,m}$:

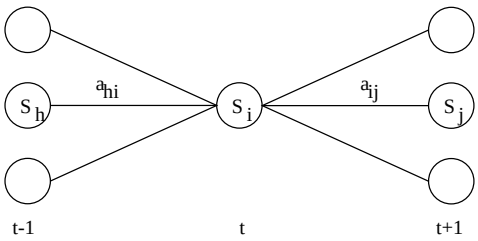


Figure 2.5: The computation of $\gamma_{s_i}(t)$, the probability of being in state $s_i$ at time $t$ while producing the sequence $x_0..x_T$. Compute the probability that the prefix of the sequence ends in $s_i$ ($\alpha_{s_i}(t, x_0..x_t)$), multiply it with the probability that the suffix of the sequence starts in $s_i$ ($\beta_{s_i}(t, x_{t+1}..x_T)$) and divide the result by the probability of the sequence ($P_h(x_0..x_T)$).

$$\begin{aligned}
\pi_{s_i} &= \textit{probability of being in state } s_i \textit{ at time } 0 = \\
&= \gamma_{s_i}(0) \qquad (2.16) \\
b_{s_i,m} &= \frac{\textit{probability of being in } s_i \textit{ while producing } m}{\textit{probability of being in state } s_i} = \\
&= \frac{\sum_t \{\gamma_{s_i}(t) \mid x_t = m\}}{\sum_t \gamma_{s_i}(t)} \qquad (2.17)
\end{aligned}$$

The numerator of equation 2.17 computes the sum of all $\gamma_{s_i}(t)$ for which $x_t = m$ holds. The summations over $t$ in equations 2.17 are necessary to take into account all $\gamma_{s_i}(t)$ ($b_{s_i,m}$ is independent of time). Apart from abling us to compute these two HMM model parameters, $\gamma_{s_i}(t)$ can also be used for computing the denominator for equation 2.9. Now we have to develop a function for computing the numerator of that equation. We start by expanding equation 2.14:

$$\begin{aligned}
P_h(x_0..x_T) &= \sum_i \alpha_{s_i}(t, x_0..x_t) * \beta_{s_i}(t, x_{t+1}..x_T) = \\
&= \sum_i \alpha_{s_i}(t, x_0..x_t) * \sum_j \beta_{s_j}(t+1, x_{t+2}..x_T) * a_{s_i,s_j} * b_{s_j,x_{t+1}} = \\
&= \sum_i \sum_j \alpha_{s_i}(t, x_0..x_t) * a_{s_i,s_j} * b_{s_j,x_{t+1}} * \beta_{s_j}(t+1, x_{t+2}..x_T)
\end{aligned} \qquad (2.18)$$

The first line is equal to equation 2.14 and in the second line we have applied the definition of $\beta_{s_i}(t, x_{t+1}..x_T)$ (equation 2.13). By reordering the elements of that line

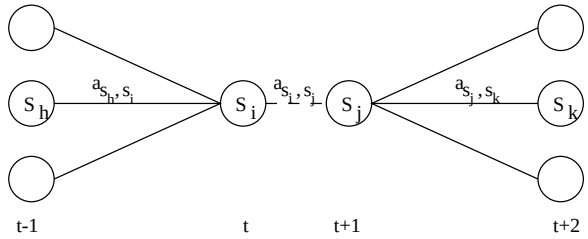


Figure 2.6: The computation of $\xi_{s_i,s_j}(t, x_{t+1})$, the probability of making a transition from state $s_i$ to $s_j$: Compute the probability that the prefix of the sequence ends in $s_i$ ($\alpha_{s_i}(t, x_0..x_t)$), multiply it with the HMM transition probability $a_{s_i,s_j}$, the HMM token production probability $b_{s_j,x_{t+1}}$ and the probability that the suffix of the sequence starts in $s_j$ ($\beta_{s_j}(t+1, x_{t+2}..x_T)$) and divide the result by the probability of the sequence ($P_h(x_0..x_T)$).

($\alpha_{s_i}(t, x_0..x_t)$ is independent of $j$) we have derived the equation at the third line. We have expanded the $\beta$-term in order to get a term within the sums in which both the computation of the probability of making the transition from $s_i$ to $s_j$ ($a_{s_i,s_j}$) and the production of token $x_{t+1}$ ($b_{s_j,x_{t+1}}$) are visible. We can use the final part of equation 2.18 for computing the probability of making a transition from one state to another (see also figure 2.6):

$$\begin{aligned} \xi_{s_i,s_j}(t, x_{t+1}) &= \textit{probability of making the transition from state } s_i \textit{ to } s_j \textit{ at time } t \textit{ and producing } x_{t+1} \\ \xi_{s_i,s_j}(t, x_{t+1}) &= \frac{\alpha_{s_i}(t, x_0..x_t) * a_{s_i,s_j} * b_{s_j,x_{t+1}} * \beta_{s_j}(t+1, x_{t+2}..x_T)}{P_h(x_0..x_T)} \end{aligned} \tag{2.19}$$

Now we need to divide $\alpha_{s_i}(t, x_0..x_t) * a_{s_i,s_j} * b_{s_j,x_{t+1}} * \beta_{s_j}(t+1, x_{t+2}..x_T)$ by the probability of producing sequence $x_0..x_T$ in order to make sure that for every $t$ the $\xi_{s_i,s_j}(t, x_{t+1})$ probabilities sum up to 1. With this $\xi_{s_i,s_j}(t, x_{t+1})$ function we are now able to compute the $a_{s_i,s_j}$ parameters of the HMM:

$$\begin{aligned} a_{s_i,s_j} &= \frac{\textit{probability of going from } s_i \textit{ to } s_j \textit{ while producing } x_{t+1}}{\textit{probability of being in state } s_i} = \\ &= \frac{\sum_t \xi_{s_i,s_j}(t, x_{t+1})}{\sum_t \gamma_{s_i}(t)} \end{aligned} \tag{2.20}$$

Again the summations over *t* are necessary to take into account all values of the functions for different *t* ($a_{s_i,s_j}$ is independent of time). Now we have obtained formulas

for computing new values for HMM parameters. Our training process will start with random values for the HMM parameters $a_{s_i,s_j}$, $b_{s_i,m}$ and $\pi_{s_i}$. We make the HMM process the training data and by using the equations 2.16, 2.17 and 2.20 we will be able to obtain better values for the HMM parameters. We continue applying this process until the probabilities that the HMM assigns to the training data become stable. At that point we hope to have obtained an HMM which is a good model for the training data.

## 2.5 Using Hidden Markov Models in practice

In this section we have introduced the mathematical background of Hidden Markov Models (HMMs). In the next two sections we will apply HMMs on training data that consists of monosyllabic Dutch words. Instead of DL-sequences the HMMs will process arbitrary sequences of characters. The HMMs will assign scores to these sequences. This scores will be equal to:

$$\begin{aligned} P_h(x_0..x_T) &= \sum_i \alpha_{s_i}(t, x_0..x_T) * \beta_{s_i}(T, e) \\ &= \sum_i \alpha_{s_i}(t, x_0..x_T) \end{aligned} \tag{2.21}$$

This is the sum for all $s_i$ of the extended forward procedure applied at the complete string of which the production ended in state $s_i$. This equation was derived from equation 2.14. $\beta_{s_i}(T)$ is equal to 1 for all $s_i$ (equation 2.12). The HMM will be trained by presenting the training data to it and applying the Baum-Welch algorithm until the HMM becomes stable. Here we have defined a stable HMM as an HMM that assigns scores to training strings that do not differ more than 1% of the scores assigned by the HMM before the final training round. In each training round the complete training data set will be processed.

When the HMM has become stable we will test it by applying it to the positive and the negative test data sets that we have described in chapter 1. The HMM should accept as many strings from the positive test data as possible and reject as many negative data as possible. We need to define a threshold score for deciding if a string is acceptable or of it is not. If a string receives a score that is higher than the threshold score it will be accepted and if it is lower than this threshold the string will be rejected.

The problem is that different HMMs will assign different scores to strings. Therefore it is impossible to determine a universal threshold value. Each HMM will require its own threshold value. Since we want all strings in the training data set to be accepted we will define the THRESHOLD SCORE as the smallest score that is assigned to an element of the training data set.

# 3 Initial Experiments

We have performed three initial experiments to find out how we need to configure the Hidden Markov Models (HMMs) in order to enable them to learn the phonotactic structure of monosyllabic Dutch words. In these experiments we have used a small data set: 3507 monosyllabic Dutch words which were extracted from the Dutch spelling guide (Spellingscommissie 1954). The HMMs were trained with 3207 words; 300 words were used as positive test data. The test data also contained an additional set of 300 randomly generated words that were constructed by taking into account the character frequencies and word length frequencies of the strings in the training data and the positive test data. No effort was taken to remove strings from the random data set that occurred in the training data or the positive test data. The data contained character representations of words rather than phonetic transcriptions. Three experiments have been performed with HMMs which contained seven states:

1. An experiment with standard data sets and random initial HMM parameter values.
2. An experiment with modified data sets and random initial HMM parameter values.
3. An experiment with modified data sets and initial HMM parameter values which had been derived from a phonological model.

We have chosen a seven-state HMM rather than an HMM with any other number of states because the linguistic model we have used for initializing the HMMs (the Cairns and Feinstein model, see section 2.4 of chapter 1) also contains 7 states. Using HMMs with the same number of states made the initialization process easier.

The next sections describe the results of these experiments.

## 3.1 A test experiment

In our first experiment we initialized the A, B and $\Pi$ matrices with random values. Then we used the Baum Welch algorithm to train the HMM. We stopped training when the scores assigned by the HMM to the words in the training set did not change more than 1% compared with the values after the previous training round. The parameters of the HMM after training can be found in figure 2.7.

In HMMs the probabilities of the outgoing transitions of each state have to sum up to 1. In this HMM this is not the fact. The reason for this is that this HMM does not handle word boundaries explicitly. For example, the outgoing transitions probabilities of the first state (top row A-matrix) sum up to 98%. This means that the probability of leaving the model after visiting state $s_1$ is 2%.

Because of this implicit handling of the word boundaries the scores assigned to words were worthless. The HMM will never assign prefixes of a sequence a score that is smaller than the sequence itself because the score of $sequence + x$ is computed

$$A = \begin{bmatrix} 0.04 & 0.94 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.51 & 0.31 & 0.00 & 0.06 & 0.02 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.15 & 0.25 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.20 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.53 & 0.00 \\ 0.37 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix} \Pi = \begin{bmatrix} 0.93 \\ 0.01 \\ 0.00 \\ 0.00 \\ 0.06 \\ 0.00 \\ 0.00 \end{bmatrix}$$

Figure 2.7: Parameter definition of the test HMM after training. The B Matrix is omitted. Apart from the backward connection from state 6 to state 1 (0.37) the HMM contains forward links only. The probabilities in the rows of the A-matrix do not add up to 1 because word boundaries were handled implicitly.

by multiplying the score of $sequence$ with the probability of $x$ (a number with 1 as maximal value). For example, *rot* will receive a larger score than *rots* and *rots* itself will receive a larger score than *rotst*. After merging the test results with the random results we discovered that, apart from two exceptions, all words of length n+1 received a smaller score than words of length n. This means that the HMM will consider impossible four-character strings more probable than valid five-character words. This clearly is wrong. Of course, in natural language it is not always the fact that a complex suffix is less improbable than a simple suffix. For example, in Dutch a syllable ending in the character *c* is more improbable than a syllable ending in *ch*. Some way has to be found to model this fact.

To handle this problem we have added an end-of-word character to all words in our training and test data. This extra character can only be the output of an eighth state in the HMM. After processing a word the HMMs have to be in this last state. This state is a so-called null state ((Van Alphen 1992)): no transitions are possible from this state. The eighth state has not been made visible in the pictures in this chapter. All transition probabilities from the other states to the last state have been put in a special vector $\Omega$ (analogous to the $\Pi$ vector, compare figure 2.7 with 2.8).

## 3.2 Orthographic data with random initialization

Our next experiment was similar to the previous apart from the fact that we have used modified data sets (with end-of-word characters). Again the A, B, $\Omega$ and $\Pi$ matrix of a seven-state HMM were initialized with random values. The Baum Welch algorithm (see section 2.4) was applied repeatedly until all training pattern scores stayed within a 1% distance of the previous scores which required 51 training rounds. The result of this was an HMM (see figure 2.9) without backward transitions (see figure 2.9, the initial HMM contained some backward transitions). The character output of the states was interesting. Here is a list of characters which are most likely to be the output of

$$A = \begin{bmatrix} 0.19 & 0.64 & 0.16 & 0.01 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.33 & 0.67 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.68 & 0.25 & 0.06 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.81 & 0.13 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.27 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.28 \end{bmatrix} \Pi = \begin{bmatrix} 0.17 \\ 0.68 \\ 0.12 \\ 0.03 \\ 0.00 \\ 0.00 \\ 0.00 \end{bmatrix} \Omega = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \\ 0.01 \\ 0.05 \\ 0.73 \\ 0.72 \end{bmatrix}$$

Figure 2.8: Parameter definition of the randomly initialized HMM after training. The HMM does not contain backward links. The elements of each row in the A-matrix together with the corresponding element in the $\Omega$-matrix add up to one.

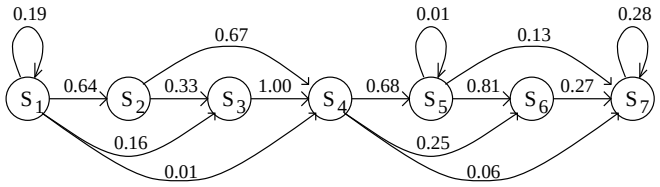


Figure 2.9: The randomly initialized HMM for monosyllabic Dutch words after training. This is a graphical representation of the $A$-matrix of figure 2.8. The HMM after training contains no backward links while before training links in any direction were possible.

the states in order of decreasing probability (ignoring characters which had less than 5% chance of being produced in the states):

- $s_1$: s c
- $s_2$: k t h p b v g d m z
- $s_3$: r l w n
- $s_4$: o a e i u
- $s_5$: e a n o u r i l j
- $s_6$: k r p l n f g m d s t
- $s_7$: s t

State $s_4$ has changed into a vowel production state. Only the five vowels *a,e,i,o* and *u* are produced by this state with a probability larger than 5%. In Dutch the *y* can be used

both as a vowel and as a consonant. In fact, *y* is the only other token that is produced by $s_4$ with a score that is larger than $10^{-8}$. The probability of producing a *j*, which is used in Dutch writing as a suffix for the *i* to create a frequent diphthong, is largest in the after-vowel state $s_5$. The initial values in the A and B matrices of the HMM were random so the HMM did a good job in discovering the difference between vowels and consonants and discovering the special position of the *j* in Dutch.

The HMM assigned the following scores for sequences in the data sets:

| | training data | positive test data | negative test data |
|---|---|---|---|
| Maximum: | $4.361*10^{-04}$ | $4.379*10^{-04}$ | $1.183*10^{-04}$ |
| Average: | $7.472*10^{-05}$ | $8.329*10^{-05}$ | $3.429*10^{-06}$ |
| Median: | $1.298*10^{-05}$ | $1.286*10^{-05}$ | $0.000*10^{+00}$ |
| Minimum: | $1.927*10^{-11}$ | $7.910*10^{-21}$ | $0.000*10^{+00}$ |

The difference between the positive test data and the negative test data is most obvious in the difference between the medians. The average score of a data set is not a good comparison value because a small number of highly probable sequences will have a large influence on this average. When we consider the median value, the score of the negative data is a lot smaller than the score of the positive data. Therefore we can say that the HMM recognized the difference between the negative data set and the positive data set.

In individual cases it is more difficult to falsify data. For example, the sequence *pajn* (score $1.971*10^{-05}$), which clearly is not a Dutch syllable, would be ranked 126th in the list of 300 test data. It receives a larger score than the perfect Dutch syllable *worp* (score $1.868*10^{-05}$).

By using our threshold definition we obtain a threshold value of $1.927*10^{-11}$, the minimum score that the HMM has assigned to an element of the training data. With this threshold value the HMM accepts 298 words of the positive test data. It rejects 2 words: *stoischt* ($2.49*10^{-12}$) and *tsjech* ($7.910*10^{-21}$). This number of rejected words is acceptable. However, the HMM also accepts 57 words of the negative test data. Among these accepted words are words which are impossible in Dutch like *jlaj* ($6.744*10^{-7}$) and *ufhf* ($3.099*10^{-11}$). This HMM does not work as we would like it to do.

## 3.3 Orthographic data with linguistic initialization

In the third experiment the initial parameter values of the HMM were derived from the syllable model defined in (Cairns and Feinstein 1982) (see section 2.4 of chapter 1). All state transitions and character productions which are possible in the Cairns and Feinstein model received a random value. The others, for example the probability of moving from $s_7$ to $s_1$ and the probability of producing a consonant in the vowel state,

$$A = \begin{bmatrix} 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.05 & 0.24 & 0.71 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.32 & 0.17 & 0.48 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.14 & 0.19 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix} \Pi = \begin{bmatrix} 0.16 \\ 0.80 \\ 0.00 \\ 0.04 \\ 0.00 \\ 0.00 \\ 0.00 \end{bmatrix} \Omega = \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \\ 0.03 \\ 0.00 \\ 0.67 \\ 1.00 \end{bmatrix}$$

Figure 2.10: Parameter definition of the linguistically initialized HMM after training. The training process has only changed the values of the non-zero entries in these matrices. The zero entries in the matrices represent impossible links that were initialized on zero. The training process could not change these values.

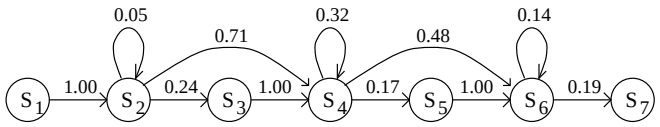


Figure 2.11: The linguistically initialized HMM for monosyllabic Dutch words after training. This is the graphical version of the A-matrix in figure 2.10. The training process only changed the weights of the links present in this picture. The result of removing specific links from the model in advance is that the HMM after training is more simple than the one in figure 2.9.

were set to zero. It was impossible for the HMM to change these zero-values. Its task was to find out the best values for the non-zero HMM parameters.

Again the Baum Welch algorithm was applied until the scores of the training patterns stayed within a 1% distance of the previous scores. This time only 15 training rounds were necessary. The parameters of the HMM after training can be found in figure 2.10. The characters which are most likely to be the output of the states are in order of decreasing probability (ignoring characters which have a probability of less than 5%):

- $s_1$: s
- $s_2$: k t h p b l v g c d
- $s_3$: r l w
- $s_4$: e a o i u

- $s_5$: n r l j m
- $s_6$: s t k p l d n f g r
- $s_7$: t s

Again, all five vowels in Dutch (*a*,*e*,*i*,*o*,*u*) are assigned to state 3. This was already the fact in the initial HMM and training did not change this fact.

The HMM assigned the following scores for sequences in the data sets:

| | training data | positive test data | negative test data |
|---|---|---|---|
| Maximum: | $9.644*10^{-04}$ | $6.794*10^{-04}$ | $1.497*10^{-04}$ |
| Average: | $7.588*10^{-05}$ | $8.634*10^{-05}$ | $3.653*10^{-06}$ |
| Median: | $6.945*10^{-06}$ | $7.654*10^{-06}$ | $0.000*10^{+00}$ |
| Minimum: | $7.418*10^{-13}$ | $6.781*10^{-13}$ | $0.000*10^{+00}$ |

A comparison between positive test data and negative test data leads to the same result as in the previous experiment: the average scores show no difference while the median scores show a clear difference between the two data sets. Comparing individual cases remains a problem. The acceptance threshold value is $7.4*10^{-13}$. With this threshold the HMM rejects one word of the positive test data set (*tsjech*). However, the HMM accepts 112 words of the negative test data among which words like *lve* ($9.202*10^{-07}$) and *fbep* ($4.111*10^{-07}$). This HMM accepts too many strings.

## 3.4 Discussion

Ideally it should be possible to choose some threshold score for an HMM and decide that every sequence with a score above this threshold is a possible sequence in the language while a sequence with a score below the threshold is not. In order to be able to do this correctly, all impossible words should receive a lower score than the 'most improbable' word in the language. However, we have not been able to find such a perfect threshold in the previous experiments. The most improbable word of the positive test data, *tsjech*, has received the score $6.781*10^{-13}$ of the linguistically initialized HMM. This score is lower than the scores of some words of the negative data set which are impossible in Dutch, for example *zrag* ($1.747*10^{-05}$), *pesf* ($7.323*10^{-06}$) and *jlaj* ($5.496*10^{-07}$). Figure 2.12 shows a comparison between words in the positive test data and words in the negative test data. The randomly generated impossible word *ddne* is about as probable as the Dutch word *snoodst*, the impossible *gvna* is about as probable as *placht* and there are more couples like that.

There are two explanations for this behavior. The first is that HMMs in general will assign a lower score to longer words than to shorter words. The Dutch word *tsjech* contains six characters while the non-words mentioned only contain four. However, this HMM feature can only be a part of the explanation. When we try finding impossible six-character words in the negative test data we find *letfdh* (score $8.218*10^{-11}$), *fobhlh*

| | | | |
|---|---|---|---|
| 281 | 0.00000002702359 | schrap | yuz |
| 282 | 0.00000002672019 | knapst | yuz |
| 283 | 0.00000002419223 | grootst | yuz |
| 284 | 0.00000002289886 | zeeuws | daxn |
| 285 | 0.00000002171036 | joodst | daxn |
| 286 | 0.00000001651132 | fijnst | tzoeh |
| 287 | 0.00000001589515 | schaak | tzoeh |
| 288 | 0.00000001508380 | blondst | tzoeh |
| 289 | 0.00000000943477 | snoodst | ddne |
| 290 | 0.00000000889425 | bruutst | yjw |
| 291 | 0.00000000858229 | schold | ywua |
| 292 | 0.00000000758883 | placht | gvna |
| 293 | 0.00000000730357 | schoor | sewz |
| 294 | 0.00000000710498 | smacht | sewz |
| 295 | 0.00000000636563 | speech | uuuar |
| 296 | 0.00000000050732 | schoolst | zgoyt |
| 297 | 0.00000000003132 | echtst | odhnf |
| 298 | 0.00000000002609 | stoischt | odhnf |
| 299 | 0.00000000001088 | knechts | qnpu |
| 300 | 0.00000000000068 | tsjech | fobhlh |

Figure 2.12: The 20 most improbable words (according to linguistically initialized HMM) in the positive test data together with negative test data that is equally probable. The scores assigned to negative strings are too high. In this respect the performance of the HMM can be improved if it can recognize that certain characters pairs do not occur in monosyllabic Dutch words: *yu*, *xn*, *dd*, *yj*, *yw*, *gv*, *wz*, *zg*, *dn*, *qn* and *lh*.

($1.03*10^{-12}$) and *edfgdg* ($6.5*10^{-13}$). These are all impossible six-character words in Dutch and only the third one receives a lower score than the valid six-character word *tsjech*.

Another explanation for the behavior of the HMM can be found when we look at the most probable processing sequence of the Dutch word *pijn* (pain) by the linguistically initialized HMM. The HMM will start in $s_2$ (p), move to $s_4$ (i), move to $s_5$ (j) and finish in $s_6$ (n). Now let's keep this state sequence and replace the characters which are produced by other characters. For example, we can replace the *i* produced in $s_4$ by an *a*. The *a* is more likely to be produced in $s_4$ than the *i* (see the state character schema in section 3.3, on each row characters are ordered from most frequent to least frequent) so we have obtained a word *pajn* which has a more probable main state sequence than the word *pijn*. Unfortunately, *pajn* is impossible in Dutch.

The problem here is clear. In monosyllabic Dutch words a *j* can follow an *i* but it cannot follow an *a*. However, the probability that the HMM will produce a *j* in $s_5$ is independent of the character produced in $s_4$. The HMM does not have memory of

previous parts of the sequence. Therefore it will still assign incorrect scores to some sequences.

A straightforward solution to this problem is to change the tokens the HMMs process from one character tokens to two or three character tokens. So we change the HMMs from a unigram model (no context characters) to a bigram model (one character context) or a trigram model (two character context). A test with a bigram HMM resulted in the desired behavior: because no *aj* bigrams occur in the training data a randomly initialized HMM will reject *pajn* (score 0.0) after training. This data format modification solves the problem of *pajn* but it might cause a new problem when we are going to work with HMMs that start from some initial phonotactic knowledge. If we want to use bigrams or trigrams, we will have to find out how to initialize a model that processes these structures. The initialization model we want to use, the Cairns and Feinstein model (see section 2.4 of chapter 1), does not contain an explicit context environment.

Another problem is that in HMMs a sequence can never be more probable than its prefix (the sequence without the final character). HMMs compute the score of a sequence consisting of a prefix plus one extra character by multiplying the score of the prefix with the probability that the character follows the prefix. Neither of these values will be larger than one. Because of this way of computation longer sequences will receive a smaller score than shorter sequences. We want to compute the probability that a sequence is present in the vocabulary of a language. This probability does not depend on the length of the sequence only. Therefore the scores the HMM compute should be changed to sequence-length-independent scores.

After having examined the scores the bigram HMM had assigned to the training data, we observed that the average scores of sequences of length n were about 10 times as large as scores of sequences of length n+1. Therefore we have decided to multiply all HMM scores with a factor $10^{sequence\ length}$ in order to decrease the influence of length on sequence score.

# 4 Experiments with bigram HMMs

In this section we will describe four series of experiments that we have carried out with bigram HMMs. We will start with a general description of the set-up of the experiments. After that we will present the results of the four experiment series. The series are divided in two groups: in one group we have used orthographic data and in the other group we have used phonetic data. In each group we have performed two series of experiments: one with HMMs that were initialized randomly and one with HMMs that we initialized by using the phonological model of Cairns and Feinstein that was described in chapter 1. Each experiment will be described in a separate section.

## 4.1 General bigram HMM experiment set-up

In order to create a bigram HMM, we modified the way the HMM interpreted strings. The unigram HMM interprets one character as one token: *splash=s-p-l-a-s-h*. Our bigram HMM divides strings into two-character tokens: *splash=sp-pl-la-as-sh*. By using this simple input interpretation, we were able to use the same theoretical learning model as in the previous experiments.

The interpretation created a problem. We consider an HMM production of a bigram as the production of the second character of the bigram in the context of the first. This means that the production of a six-character word like *splash*$\star$ ($\star$ is the word-end character) contains six steps (the five bigrams mentioned above plus the bigram *h*$\star$). These six steps produce the characters *plash*$\star$. The first character of the word will never be produced because during word production there is no bigram which contains the first character at the second position.

The omission of the production of the first character of the words generated erratic behavior from the HMMs. The one-character words in the training set received a zero-score from the HMMs which caused them to collapse. We solved this problem by expanding our data representation by adding a word-start character to all words. The production of a word then involved the production of a word-start character/first character bigram (for *splash* this is $\wedge s$, $\wedge$ is the word-start character) which means that now the first character of the word will be produced.

Apart from changing the representation of the data, we made another change in these bigram experiments. We observed that the random initialization of the HMM parameters influences the HMM performance. Because of the initial values of the model, the learning performance can differ. To minimize this influence we performed have five experiments with different initial values in each series. The average performance of the five experiments has been used as the result of the experiment series.

In these experiments we have used the large data set described in section 2.2 of chapter 1: 5577 words in orthographic representation or 5084 words in phonetic representation as training data, a 600 words positive test data set and a negative test data set containing 600 strings.

## 4.2 Orthographic data with random initialization

The first series of five experiments involved training randomly initialized HMMs on orthographic data. We have used the orthographic data described in section 2.2 of chapter 1: a training data set of 5577 monosyllabic Dutch words, a positive test data set of 600 monosyllabic Dutch words that did not occur in the training data set and a set of 600 negative test strings that did not occur in the previous two data sets. The results of these experiments can be found in figure 2.13.

On average the HMM needed 77.8 rounds to become stable. The stability criterion used was the same as in the earlier experiments: the HMM was considered stable when after a training round the evaluation scores of the words in the training sequence remained within a 1% distance from the scores that were assigned to them before this

| nbr. | rounds | threshold | positive accepts | negative rejects |
|---|---|---|---|---|
| 1. | 108 | $3.599*10^{-13}$ | 591 | 564 |
| 2. | 87 | $1.090*10^{-12}$ | 594 | 535 |
| 3. | 81 | $5.909*10^{-14}$ | 594 | 544 |
| 4. | 48 | $1.311*10^{-13}$ | 594 | 523 |
| 5. | 65 | $5.637*10^{-13}$ | 594 | 565 |
| avg. | 77.8±20.3 | | 593.4±1.2 (98.9%) | 546.2±16.4 (91.0%) |

Figure 2.13: The results of five experiments with randomly initialized bigram HMMs that processed orthographic data. After an average of 78 training rounds the HMMs accepted on average 593 words of the positive test data set (98.9%) and rejected 546 strings of the negative test data set (91.0%). Nineteen negative strings were accepted by all five HMMs and five positive words were rejected by all five HMMs.

round. The resulting models accepted on average 593.4 of the 600 positive test words (98.9%) and rejected 546.2 of the 600 negative strings (91.0%). Six positive test words were rejected by all models: *ij's*, *q's*, *fjord*, *f's*, *schwung* and *t's*. The models assigned a low score to *ij's* because it contains a trigram (*ij'*) that does not occur in the training data. The other five words contain a bigram that was not present in the training data. The HMMs set the probability of occurrence of this bigram to zero. Therefore the scores of these words also became zero.

Nineteen of the 600 negative strings were accepted by all five models: *deess*, *enc*, *horet*, *ieer*, *maung*, *metet*, *oarp*, *ooe*, *oui*, *ousc*, *sassk*, *sspt*, *teaq*, *tskip*, *tspt*, *uai*, *uast*, *waese* and *woic*. These strings are not acceptable as monosyllabic Dutch words. Some of them consists of two syllables (*horet* and *metet*) and others do not even contain a vowel (*sspt* and *tspt*). Most of these misclassifications of the models can be explained by the small context the models have been using. For example, *ieer* consists of three very common bigrams *ie*, *ee* and *er* and the models use this fact to assign a high score to the word. However, the combination of these three bigrams in a Dutch monosyllabic word is not possible. The models could have been prevented from making this mistake if they had been using a larger context: the trigram *iee* does not occur in the training data. The two-syllable strings in this set can be explained by the occurrence of some accepted words from foreign languages in the training data like *shaket* and *faket*. Acceptance of the consonant words was caused by the presence of the two consonant interjections *pst* and *sst* in the training data set.

The errors for the positive test set are reasonable but we are less satisfied with the errors the HMMs make for the negative test data. The tendency of the models to accept too many unacceptable strings can be contributed to the small one character context that they are using. Expanding the context of the models would mean using trigrams instead of bigrams. However, then we would run into computational problems. The trigram models will simply need more computational resources for training than we presently have available. Therefore we will try to improve the performance of these

$$A = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \Pi = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \Omega = \begin{bmatrix} 0.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

$$B = \begin{bmatrix} vowels & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ consonants & 1.0 & 1.0 & 1.0 & 0.0 & 1.0 & 1.0 & 1.0 \\ y & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

Figure 2.14: Initial configuration for bigram HMMs for orthographic data that start learning from linguistic knowledge. The value 0.0 indicates an impossible link or an impossible character output and the value 1.0 indicates possible links or character output.

models in a different fashion. We will supply the HMMs with some initial linguistic knowledge and thus attempt to put constraints on the models that will be produced by training. Our hope is that the constrained models will be more strict when evaluating negative strings.

## 4.3 Orthographic data with linguistic initialization

In the second series of experiments we have used the bigram HMMs with an initial configuration that was derived from the model from Cairns and Feinstein. In the initial configuration we ignored the first characters of bigrams and we treated the output of a *XY* bigram as the output of a *Y* unigram. The initial configuration contained two types of values: value 0.0 and values larger than 0.0. The first value type indicated links or bigram outputs which are impossible according to the phonological model from Cairns and Feinstein. It was impossible for the HMM to change this value during training. The other value type was used for parameters that represented possible links or possible bigram outputs. These parameters were initialized with a random value.

An outline of the initial HMM configuration can be found in figure 2.14 and figure 2.15. This initial configuration was based on the Cairns and Feinstein model (see section 2.4 of chapter 1). The original version of the Cairns and Feinstein model is unable to explain the structure of all strings in orthographic training data. An HMM that would use this model as initialization model would assign zero-scores to part of our training data. This would make it unfit as an orthographic model since strings with zero-scores should be rejected and we require that our models accept all training data.

We have made three extensions to the standard Cairns and Feinstein model in order to make it usable as an initial orthographic model. First, the initial configuration

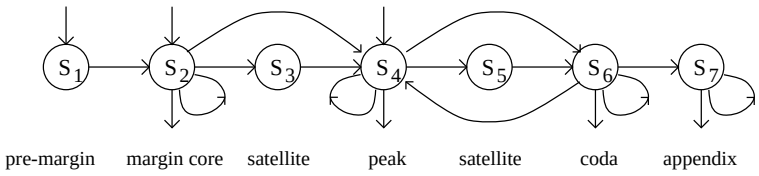


Figure 2.15: Initial bigram HMM for orthographic data. This is a graphical representation of the HMM parameters presented in figure 2.14. The visible features that were added to the original Cairns and Feinstein model are the self-links at state 2 (margin core), state 4 (peak) and state 6 (coda), the outward link from state 2 (margin core) and the backward link from state 6 (coda) to state 4 (peak). These links are necessary to enable the model to handle consonant clusters (*ch*, vowel clusters (diphtongs) and the non-vowel words in the training data set like *b*, *c*, *pst* etcetera.

for the A-matrix contains three extra links from a state to itself namely for state 2 (margin core), state 4 (peak) and state 6 (coda). These links are necessary because in orthographic data some consonants (for example *ch*) and some diphtongs (for example *au*) are represented by two tokens while the production of specific tokens is restricted to specific states by the B-matrix, for example vowels can only be produced by state 4 (peak). When sounds are represented by a cluster of tokens it is necessary to use a state that should produce such a cluster a number of times in succession. In order to be able to do that the state should contain a link to itself. The link from state 7 (appendix) to itself is necessary because the appendix can contain more than one *s* or *t*, for example: *tam*, *tams* and *tamst* in which the *s*'s and *t*'s should appear in the appendix (this is already a feature of the standard Cairns and Feinstein model).

The second extension is the backward link: state 6 $\rightarrow$ state 4 = coda $\rightarrow$ peak. The HMMs were not able to process the training data set without this backward link. Leaving out the backward link would make the HMMs assign zero-scores to accepted loan words as *ace*, *creme* and *file*. These words contain two isolated vowel groups. However, we will enable only one state (peak) to produce vowels. To be able to process words with two isolated vowel groups the HMMs will have to use this state twice and therefore a backward link to state 4 (peak) is necessary. The danger of having such a link in the HMMs is that they could use it for assigning high scores to multiple syllable words.

The third extension of the Cairns and Feinstein model in this initial model is the added possibility to finish a string after having processed the margin core (state 2). In the initial model this is represented by a link from state two to the hidden eight state. This link becomes visible in the second element of $\Omega$-matrix in figure 2.14. The link is necessary to enable the HMMs to process interjections like *pst* and *sst* and the consonants of alphabet (*b*, *c*, *d*, etc.) that are also present in our training data set as words.

| nbr. | rounds | threshold | positive accepts | negative rejects |
|---|---|---|---|---|
| 1. | 43 | $3.313*10^{-15}$ | 595 | 509 |
| 2. | 17 | $9.947*10^{-17}$ | 595 | 511 |
| 3. | 63 | $3.337*10^{-15}$ | 595 | 508 |
| 4. | 22 | $1.036*10^{-16}$ | 595 | 513 |
| 5. | 47 | $1.439*10^{-14}$ | 595 | 514 |
| avg. | 38.4±16.9 | | 595.0±0.0 (99.2%) | 511.0±2.3 (85.2%) |

Figure 2.16: The results of five experiments with linguistically initialized bigram HMMs that processed orthographic data. After an average of 38 training rounds the HMMs accepted 595 words of the positive test data set (99.2%) and rejected on average 511 strings of the negative test data set (85.2%). Eighty-two negative strings were accepted by all five HMMs and five test words were rejected by all five HMMs.

The initial B-matrix contains three groups of tokens. The first group consists of the vowels which can only be produced by state 4 (peak). The second group consists of the consonants. These can be produced by any state except state 4 (peak). We have regarded the single quote character (', among others present in *d'r*, *j's* and *vla's*) as a vowel. Finally there is the *y* which can be used both as a vowel and a consonant in Dutch. This character can be produced by any state.

Like in the previous section we performed five experiments with different initial parameter values. The results can be found in figure 2.16. The HMMs needed on average only 38.4 rounds to become stable. Thus they trained faster than the HMMs that were initialized randomly ($t(4)=3.0$, $p<0.025$, see section 2.5 of chapter 1). The linguistic initialization procedure resulted in small increase of the positive test words that were accepted: on average 595.0 compared with 593.4 for the randomly initialized HMMs ($t(4)=2.7$, $p<0.05$). Contrary to our goal the HMMs with linguistic initialization rejected fewer incorrect strings from the negative test data set than the HMMs that were initialized randomly: 511.0 compared with 546.2 ($t(4)=4.3$, $p<0.01$). We have to conclude that the phonological model we used for initializing the HMMs is not suitable for our orthographic data.

We have inspected one of the HMMs that resulted after training from a random initialization. This model suggested that we should make three changes to our initial model. First, we should allow vowel production in two states instead of one state. The model suggested to use state 6 as an extra vowel state. This state would be allowed to produce both consonants and vowels. The extra vowel state is necessary for being able to process the foreign words with two vowel clusters. As a result of extending the production capabilities of state 6 we can remove the backward link between state 6 and state 4. This is the second change we make to the model. Finally, the trained HMM processed the quote character as a consonant, not as a vowel. We will make this change in the initial model as well. There were other differences between the trained

$$A = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \Pi = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \Omega = \begin{bmatrix} 0.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

$$B = \begin{bmatrix} vowels & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 1.0 & 0.0 \\ consonants & 1.0 & 1.0 & 1.0 & 0.0 & 1.0 & 1.0 & 1.0 \\ y & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

Figure 2.17: Modified initial configuration for bigram HMMs for orthographic data that start learning from linguistic knowledge. In order to mimic the behavior of the trained models with random initialization, we have removed the backward link from state 6 to state 4 and allowed state 6 to produce vowels. Furthermore, we have treated the quote character ' as a consonant instead of a vowel.

randomly initialized model and our linguistic initial model but these three were the most important ones. The initial model can be found in figure 2.17 and 2.18.

The HMMs trained with this initial configuration performed better with respect to the negative strings than the previous linguistically initialized HMMs (figure 2.19). They rejected 567 of the 600 negative strings (94.5%) compared with on average 511.0 rejected strings by the previous HMMs ($t(4)=49.1$, $p<0.005$) The training time needed was about as long as the previous HMMs (52.2 rounds compared with 38.4 rounds, $t(4)=1.1$ $p>0.1$) while they accepted fewer strings of the positive test data (593.4 compared with 595, $t(4)=6.5$, $p<0.005$). If we compare these HMMs with the randomly initialized HMMs we find out that they need about the same training time (52.2 rounds compared with 77.8 rounds, $t(4)=1.8$, $p>0.05$), accept the same number of positive test words (593.4 compared with 593.4, $t(4)=1.8$, $p>0.25$) and reject more strings of the negative test data set (567 compared with 546.2, $t(4)=2.5$, $p<0.05$).

We can conclude that for orthographic data the performance of the HMMs can be improved by starting training from a good initial HMM configuration. Constructing the initial HMM from a phonological model without making any data-specific adjustments did not provide us with good results. The difference between a good phonological model and a good orthographic model is too large.

## 4.4 Phonetic data with random initialization

The third series of five experiments involved training randomly initialized HMMs to process phonetic data. We have used the phonetic data described in section 2.2: a

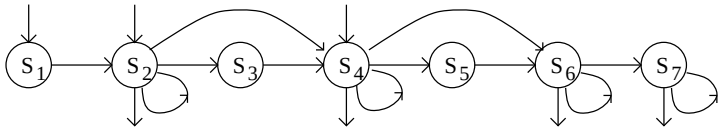


Figure 2.18: Modified initial bigram HMM for orthographic data. This is a graphical representation of the HMM parameters presented in figure 2.17. The backward link from state 6 to state 4 has been removed from the model. The other features that were added to the original Cairns and Feinstein model, the self-links at state 2 (margin core), state 4 (peak) and state 6 (coda), and the outward link from state 2 (margin core), remained in the model.

| nbr. | rounds | threshold | positive accepts | negative rejects |
|---|---|---|---|---|
| 1. | 55 | $2.841*10^{-14}$ | 593 | 567 |
| 2. | 52 | $5.974*10^{-14}$ | 594 | 567 |
| 3. | 21 | $6.092*10^{-14}$ | 594 | 567 |
| 4. | 82 | $2.778*10^{-14}$ | 593 | 567 |
| 5. | 51 | $2.847*10^{-14}$ | 593 | 567 |
| avg. | 52.2±19.3 | | 593.4±0.5 (98.9%) | 567±0.0 (94.5%) |

Figure 2.19: The results of five experiments with bigram HMMs that processed orthographic data and used the modified linguistic initialization. After an average of 52 training rounds the HMMs accepted 593 words of the positive test data set (98.9%) and rejected 567 strings of the negative test data set (94.5%). 33 negative strings were accepted by all five HMMs and 6 test words were rejected by all five HMMs.

training data set of 5084 monosyllabic Dutch words, a positive test data set of 600 monosyllabic Dutch words that did not occur in the training data set and a negative test set of 600 strings. The results of these experiments can be found in figure 2.20.

These five HMMs performed equally well as the five randomly initialized HMMs that were trained on the orthographic data. They needed on average 68.6 training rounds to become stable (similar the 77.8 for orthographic data, $t(4)=0.5$, $p>0.25$) after which they accepted on average 594.6 words (99.1%) of the positive test data set (similar to the 593.4 for orthographic data, $t(4)=1.7$, $p>0.05$) and rejected 565.6 strings (94.3%) of the negative test data set (better than the 546.2 for orthographic data, $t(4)=2.4$, $p<0.05$). Five words of the positive test data set were rejected by all five HMMs: fjord [fjɔrt], fuut [fyt], schwung [ʃwuŋ], schmink [ʃmiŋk] and schminkt [ʃmiŋkt]. These words contain bigrams that are not present in the training data: [fj], [fy], [uŋ] and [iŋ]. The HMMs assign the score 0 to these bigrams and therefore the scores of these words also become 0.

| nbr. | rounds | threshold | positive accepts | negative rejects |
|---|---|---|---|---|
| 1. | 48 | $3.401*10^{-10}$ | 595 | 566 |
| 2. | 58 | $6.274*10^{-11}$ | 595 | 568 |
| 3. | 131 | $2.151*10^{-10}$ | 593 | 563 |
| 4. | 41 | $4.240*10^{-10}$ | 595 | 565 |
| 5. | 65 | $4.425*10^{-10}$ | 595 | 566 |
| avg. | 68.6±32.3 | | 594.6±0.8 (99.1%) | 565.6±1.6 (94.3%) |

Figure 2.20: The results of five experiments with randomly initialized bigram HMMs that processed phonetic data. After an average of 69 training rounds the HMMs accepted on average 595 words of the positive test data set (99.1%) and rejected 566 strings of the negative test data set (94.3%). Twenty-eight negative strings were accepted by all five HMMs and five positive test words were rejected by all five HMMs. Twenty-four of the universally accepted negative strings had an acceptable phonetic representation. When we take this into account, the average performance of the HMMs on the negative test data set becomes 98.3%.

Twenty-eight negative strings were accepted by all five HMMs: astt [ɑst], brhat [brɑt], cci [ki:], ckeds [skɛts], cto [sto], deess [de:s], ejh [ɛj], ejss [ɛjs], fovhst [fɔfst], hurwd [hœrwt], kkraeb [kre:p], klolc [klɔlk], kuktzt [kœktst], nalc [nɑlk], oarp [o:rp], ousc [ɑusk], piuttd [pju:t], roqks [rɔks], sassk [sɑsk], teaq [ti:k], terh [tɛr], tskip [tskɪp] ttik [tɪk], ttra [tra:], ttui [tʌy], twosd [twɔst], udsb [œtsp] and uzs [y:s]. The orthographic representations of these words are not acceptable as monosyllabic Dutch words but 24 of the 28 of the phonetic representations are acceptable. The four strings that do not have an acceptable phonetic representation are: hurwd [hœrwt], klolc [klɔlk], tskip [tskɪp] and udsb [œtsp]. The conversion of the orthographic representations to the phonetic representations has been done by a native Dutch speaker and this might have resulted in 'quasi-Dutch' transcriptions. An inspection of the negative data set resulted in two more acceptable transcriptions: tzips [tsIps] and ttsue [tsy:]. If we omit the 26 acceptable negative strings from the data, the model has rejected on average 564.2[3] of 574 negative strings (98.3%) which is a good score.

## 4.5 Phonetic data with linguistic initialization

The results of the randomly initialized HMMs that processed phonetic data were acceptable. Still we are interested what the HMMs will do when they are provided with basic initial phonotactic knowledge. In this experiment series we will apply linguis-

[3]When the strings tzips [tsIps] and ttsue [tsy:] are removed from the data the average rejection scores will decrease with 1.0 because all HMMs rejected ttsue [tsy:] and with another 0.4 because two HMMs rejected tzips [tsIps].

$$A = \begin{bmatrix} 1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \Pi = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \quad \Omega = \begin{bmatrix} 0.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 0.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

$$B = \begin{bmatrix} vowels & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ consonants & 1.0 & 1.0 & 1.0 & 0.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

Figure 2.21: Initial configuration for bigram HMMs for phonetic data that start from basic phonotactic knowledge. The value 0.0 indicates an impossible link or an impossible character production and the value 1.0 indicates possible links or possible character productions. The non-zero parameters were initialized with random values before training.

tically initialized HMMs to phonotactic knowledge. The initialization model we will use is the model of Cairns and Feinstein we have introduced in section 2.4 of chapter 1. Again we had to adapt the standard Cairns and Feinstein model in order to enable it to handle all words of the training data set. This set contains two interjections (*pst* and *sst*) and one abbreviation (*s*) that cannot be explained with the Cairns and Feinstein model. In order to enable the models to handle these words we have added a link from state 2 (margin core) to hidden eight state (see $\Omega$-matrix in figure 2.21 and figure 2.22). This link will enable the model to accept strings that do not contain a vowel. Furthermore, we had to link state 1 (pre-margin) to itself in order to enable the models to process the consonant clusters in *pst* and *sst.* All impossible links were removed from the HMMs and they were prevented from restoring them. The initial model can be found in figures 2.21 and 2.22.

Like in the experiments with the orthographic data the program was supplied with information about the difference between vowels and consonants. Vowels were allowed as the output of state 4 (peak) only and consonants were allowed as the output of any other state (see B-matrix in figure 2.21). The task of the HMMs was to find out the best values of the state transitions in the model (A-matrix, $\Pi$-matrix and $\Omega$-matrix) and discover what consonants are allowed in which state (B-matrix). The values of impossible links (A-matrix, $\Pi$-matrix and $\Omega$-matrix) and probabilities of impossible productions of characters were initialized with zero. All other model parameters were initialized with a random value. We performed five experiments with this set-up. The results can be found in figure 2.23.

The HMMs needed on average 28.2 rounds to become stable. Thus they need less training rounds than the models without linguistic initialization (68.6 rounds, t(4)=2.3,

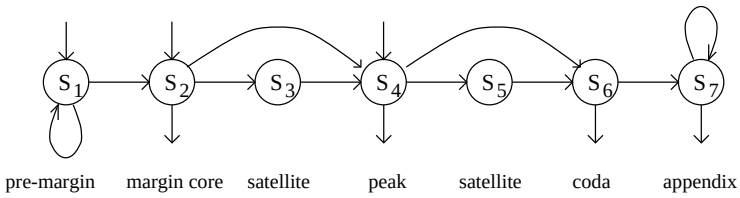


Figure 2.22: Initial bigram HMM for phonetic data. This is a graphical representation of the HMM parameters presented in figure 2.18. The two visible differences between this model and the original phonological model of Cairns and Feinstein are the extra link leaving state 2 (margin core) and the self-link at state 1 (pre-margin). These links are necessary to enable the model to handle the non-vowel interjections *pst* and *sst*, the abbreviation *s* and the consonant clusters in these strings.

| nbr. | rounds | threshold | positive accepts | negative rejects |
|---|---|---|---|---|
| 1. | 33 | $1.575*10^{-10}$ | 594 | 570 |
| 2. | 48 | $1.614*10^{-10}$ | 595 | 570 |
| 3. | 17 | $1.213*10^{-10}$ | 595 | 570 |
| 4. | 10 | $3.562*10^{-11}$ | 595 | 569 |
| 5. | 33 | $3.562*10^{-11}$ | 594 | 570 |
| avg. | 28.2±13.4 | | 594.6±0.5 (99.1%) | 569.8±0.4 (95.0%) |

Figure 2.23: The results of five experiments with linguistically initialized bigram HMMs that processed phonetic data. After an average of 28 training rounds the HMMs accepted on average 595 words of the positive test data set (99.1%) and rejected 570 strings of the negative test data set (95.0%). When we removed strings with an acceptable phonetic transcription from the negative test data the HMMs obtained a rejection score of 99.1% on this test set.

$p<0.05$). The models accepted on average 594.6 words of the positive test data and this is exactly as many as the HMMs with random initialization (594.6 words, $t(4)=0.0$, $p>0.25$). They rejected more strings of the negative test data set (569.8) than the models of the previous section (565.6 strings, $t(4)=5.0$, $p<0.005$).

Twenty-nine negative strings were accepted by all five HMMs and five test words were rejected by all five HMMs. The five uniformly rejected test words were the same as in the experiments with randomly initialized HMMs. Again the words received the score zero because they contained bigrams that do not occur in the training data. Of the 29 accepted strings from the negative test data set 24 were in the set of 26 reasonable phonetic transcriptions (see previous section). The other five universally accepted strings were not acceptable: ephtsb [ɛptsp], hurwd [hœrwt], klolc [klɔlk],

sfhi [sfi:] and udsb [œtsp]. When we remove the 26 reasonable transcriptions from the data the HMM has rejected on average 568.6 of 574 negative strings (99.1%). This is an improvement compared with the average score of 98.3% achieved in the experiments with random initialization ($t(4)=4.9$, $p<0.005$).

# 5 Concluding remarks

It is possible to use Hidden Markov Models (HMMs) for building phonotactic models from a list of monosyllabic words. The resulting HMMs show recognition of language specific features such as vowel-consonant distinction. Unigram HMMs perform well in recognizing the difference between a set of positive test data and a set of negative test data. However, recognizing the difference between individual cases is a problem: the unigram HMMs often assign higher scores to incorrect data than to correct language data. The main problems are that standard unigram HMMs do not pay attention to the context of a character and that they exaggerate the influence of sequence length on sequence score. Providing the unigram HMMs with initial phonotactic knowledge shortens the training phase but it does not increase their performance after training.

Bigram HMMs with score correction for sequence length perform better. They misclassify few words and recognize a clear difference between positive test data and negative test data accepting on average 99.1% of a set of unseen correct words in phonetic representation (98.9% for orthographic data) while rejecting on average 98.3% of a set of impossible negative phonetic test sequences (91.0% for orthographic data). Providing initial linguistic knowledge to the HMMs caused a significant and large increase of the training speed for the phonetic data but only small increases in performance. The number of training rounds needed went down from an average of 68.6 to 28.2 for phonetic data ($t(4)=2.3$, $p<0.05$, see section 2.5 of chapter 1). The acceptance rate for the positive phonetic test data was the same for randomly and linguistically initialized HMMs (99.1%) but the rejection rate of the negative test data showed a small increase from 98.3% for the randomly initialized HMMs to 99.1% for the linguistically initialized HMMs ($t(4)=6.2$, $p<0.005$). HMMs with initial linguistic knowledge that processed orthographic data needed approximately the same number of training rounds as HMMs that were initialized randomly and performed worse. The phonological model that we used for initializing the HMMs was not suitable for orthographic data.

We can examine two of the three research questions mentioned in chapter 1. The phonetic data format seems to be better suitable for our problem. HMMs that processed orthographic data accepted as many positive test words as those that processed phonetic data but the latter rejected significantly more negative strings. Starting from basic phonotactic knowledge enables the HMMs to produce better models but the difference was only noticeable in the rejection rates of the negative data. Both HMMs that processed orthographic data as those that processed phonetic data accepted as many positive test data with and without initial knowledge but the rejection rates for negative data were significantly larger for the initialized HMMs.

The models that we have built in this chapter suffer from one of the problems that were mentioned in (Fudge et al. 1998) namely the presence of loan words in the data. These words have complicated the models. However, it is not easy to remove the loan words from the data set without making assumptions about the structure of the words. Therefore, we have chosen to leave these words in the training and test data. Fudge and Shockey have also recognized the problem of accepting incorrect strings when no context information is used. We have discussed this problem in section 3.4 with the example string *pajn* and solved the problem by using bigram HMMs instead of unigram HMMs. Our orthographic bigram models reject two of the three problematic words mentioned in (Fudge et al. 1998): *smlasr* and *sdring*. These strings should also be impossible in Dutch. The third word, *ssming*, is accepted by the models because the unusual onset *ss* occurs in the interjection *sst* which is present in the training data.

Fudge and Shockey also discuss the difference between accidental and systematic gaps in language patterns. Our approach to this is to regard any string that is not accepted by a model as a systematic gap and regard all strings that are accepted by a model but that are not present in the language as an accidental gap. Thus the difference between accidental and systematic gaps has become a model-dependent difference.

The models derived in this chapter satisfy two of the five properties Mark Ellison outlined in his thesis (Ellison 1992). They are cipher-independent (independent of the symbols chosen for the phonemes) and language-independent (they make no initial assumptions specific for a certain language), but their internal structure is neither accessible nor linguistically meaningful. The HMMs also fail to satisfy Ellison's first property (operation in isolation) because they receive preprocessed language input: monosyllabic words. The removal of the monosyllabicity constraint we put on our training data is an interesting topic for future work.