Bachelorarbeit

Hidden Markov models

Tobias Nelsen
18. Juni 2013

Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl für Grundlagen der Programmierung

Betreuer: Dr. rer. nat. Torsten Stüber
Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler

Contents

1 Introduction

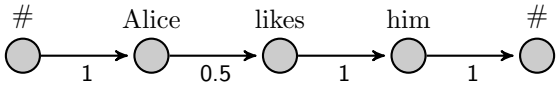
One of the concerns of natural language processing is the quality valuation of a sentence. Information which can be used for this purpose consist in form of a corpus, a collection of texts. A set of novels or speeches delivered in the European Parliament may serve as an example for such a corpus. As the corpus is created by humans, it can be assumed that the sentences within the corpus are of good quality and therefore a suitable basis for the valuation. A small corpus could consist of two sentences and look as follows:

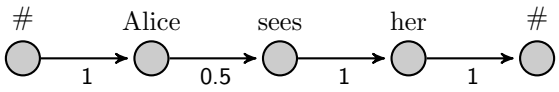
Alice likes him Alice sees her

One way to use the corpus is by counting how many times the sentence that is to be valuated occurs in the corpus. If it does so frequently, it is probably of high quality.
This approach has two disadvantages: On the one hand, for a large corpus there is an enormous amount of memory required as the whole corpus respectively one occurrence of each sentence of the corpus together with its frequency has to be stored. On the other hand, the problem of *overfitting* [St3] occurs - the quality valuation is limited exclusively to the corpus: A sentence which to human knowledge is of high quality, but does not appear in the corpus and differs merely slightly from one sentence in the corpus, for example by using another verb, would get rated badly.
One possibility to encounter these problems is not to store the frequency of the whole sentence of the corpus but to store the frequency of certain parts of it and to use these for the valuation. This leads to a reduced amount of space needed as only all possible fragments of a sentence (the length of these fragments is restricted) and its frequencies have to be stored. Furthermore, a sentence that is to be rated, does not have to be part of the corpus itself. It is now sufficient that the corpus contains the fragments of the sentence, possibly split across multiple different sentences in the corpus. This allows a variety of sentence to be rated better.
A model which realizes this approach is the *bigram model* [Vog12]. The bigram model consists of a conditional probability distribution b which specifies the probability that given a specific word, this word is followed by another specific word in a sentence. To extract the conditional probability distribution for the corpus above, the method of *conditional relative frequency* can be used. At this the probability for a given a word v_1 to be followed by a word v_2 is calculated by counting the occurrences of v_1v_2 in the corpus and dividing this by the number of occurrences of v_1 in the corpus. Hence the conditional probability distribution reads as follows:

$b(. \mid .)$	#	Alice	likes	sees	him	her
#	0	0	0	0	1	1
Alice	1	0	0	0	0	0
likes	0	0.5	0	0	0	0
sees	0	0.5	0	0	0	0
him	0	0	1	0	0	0
her	0	0	0	1	0	0

marks the start and end of a sentence. Note that summing over the probabilities in a column yields 1.
Applying the notion of a *stochastic process* from [MS99, p. 45], the bigram model can be considered as a stochastic process which generates sentences according to a probability distribution. Generating a sentence respectively performing a *random walk* in the stochastic process based on a bigram model works as follows: Starting with the # symbol, according to the conditional probability distribution b a word is selected randomly. In the same manner this procedure is repeated with the selected word to choose the next word following in the sentence. This way of proceeding is done until the # symbol is reached again.
For the conditional probability distribution above only two sentences can be generated, those in the corpus above:





The probability of a sentence is therefore calculated by multiplying the probabilities of transitioning between neighboring words in the sentence as described before.
A remaining problem of the bigram model is the problem of *sparse data* [MS99, pp. 198-199]. According to this, even for huge corpora there are meaningful sentences where some neighboring words of these sentences are not comprised in the corpora. Therefore, transitioning from the first to the second of these words and thus the sentences consisting of these neighboring words are assigned a probability of zero.
For example consider the sentence "Alice likes Alice". Because "likes Alice" does not appear in the corpus, $b(\text{likes} \mid \text{Alice}) = 0$ and therefore the probability of the sentence is zero.

One approach to describe the grammar behind the corpus better and thereby to solve the problem above is the use of the hidden Markov model [BPSW70]. It comprises the idea of considering the words of a sentence as observations, whereas each observation is evoked by a hidden state. Hidden states can be imagined as syntactic categories such as nouns, verbs and adjectives [MS99, p. 81]. In this case the bigram model is not used with words, but with syntactic categories.
Referring to the example above, this abstraction shall cause that "Alice likes Alice" is rated with a high probability, since "Alice" could be thought of being produced by the same state as "him" or "her", for example by the state "noun".

In the following thesis, the definition of the hidden Markov model and solutions to two major problems, calculating the probability of a sentence and finding a suitable hidden Markov model for a given corpus, are presented. The latter problem is solved by using the *Baum-Welch algorithm* [BPSW70]. Furthermore, an implementation of hidden Markov models is described and the quality of the bigram model and hidden Markov model is compared empirically. Finally, two further applications of hidden Markov models in natural language processing are described.

2 Preliminaries

The following definitions, axioms, lemmas and theorems are obtained from [St3].

Definition 2.1: For every state of the world, a *random variable* $\mathbb{X}$ *over a set* X is assigned exactly one value of X.

Let $x \in X$ and $Y \subseteq X$.
Then $\mathbb{X} = x$ refers to the world being in a state at which $\mathbb{X}$ is assigned the value x and $\mathbb{X} \in Y$ refers to the world being in a state at which $\mathbb{X}$ is assigned a value of Y.

Let $\mathbb{X}_1$ be a random variable over X_1, $\mathbb{X}_2$ a random variable over X_2, and ${X_1}' \subseteq X_1$, ${X_2}' \subseteq X_2$.
Then $\mathbb{X}_1 \in {X_1}', \mathbb{X}_2 \in {X_n}'$ refers to the world being in a state in which both $\mathbb{X}_1 \in {X_1}'$ and $\mathbb{X}_2 \in {X_2}'$ hold. □

Definition 2.2: Let $\mathbb{X}$ be a random variable over X and $x \in X$.
An event s represents a set of states of the world and is defined as follows using EBNF:

$$s ::= \mathbb{X} = x \mid \mathbb{X} \neq x \mid \mathbb{X} \in X \mid \mathbb{X} \notin X \mid \text{true} \mid s, s$$

whereas *true* is an event which holds for every state of the world. □

Axiom 2.3: Let s be an event, $\mathbb{X}$ a random variable over X, $X' \subseteq X$ and $(X_i \mid i \in I)$ a countable partition of X'.
The probability of set of states of the world satisfies the following conditions:

$$P(true) = 1$$
$$P(\mathbb{X} \in X', s) = \sum_{i \in I} P(\mathbb{X} \in X_i, s)$$

□

Axiom 2.4: Let s and s' be events. If s and s' represent the same set of states of the world, s and s' are called *equivalent*, which is denoted as

$$s \equiv s'.$$

□

Lemma 2.5: Let s and s' be events. If $s \equiv s'$ then

$$P(s) = P(s').$$

□

Definition 2.6: Let s and s' be events and $P(s') > 0$. Then the *conditional probability of s given s'* is defined as

$$P(s \mid s') = \frac{P(s, s')}{P(s')}.$$

□

Theorem 2.7: Let $\mathbb{X}$ be a random variable over the countable set X and s an event. The *law of total probability* states that

$$P(s) = \sum_{x \in X} P(\mathbb{X} = x, s).$$

□

The following definitions are obtained from [Vog12].

Definition 2.8: Let X be a countable set.
Then $p : X \to [0, 1]$ with $\sum_{x \in X} p(x) = 1$ is called a *probability distribution over* X.
The set of all probability distribution over X, $\mathcal{M}(X)$, is called *probability model on* X.
The function $h : X \to \mathbb{R}_{\geq 0}$ is called *corpus over* X.
The *supply of* h is defined as $supp(h) = \{x \in X \mid h(x) \neq 0\}$ □

Definition 2.9: Let X and Y be countable sets.
A *conditional probability distribution* over X given Y is the function $p : Y \to \mathcal{M}(X)$. To simplify the notation, $p(x \mid y)$ instead of $(p(y))(x)$ is written. □

Definition 2.10: Let $\mathcal{M} \subseteq \mathcal{M}(X)$ and h be a corpus over X.
Then the *maximum likelihood estimate of* $\mathcal{M}$ *on* h is defined as

$$mle(h, \mathcal{M}) = \arg\max_{p \in \mathcal{M}} L(h, p)$$

whereas $L(h, p)$ is called Likelihood of h under p and is defined as

$$L(h, p) = \prod_{x \in X} p(x)^{h(x)}.$$

□

3 Hidden Markov models

Many different versions of hidden Markov models exist in the literature. Exemplarily, there are versions of hidden Markov models, in which there is no final state [Rab89] and in which an observation depends on multiple hidden states [MS99, p. 324].
To achieve that the sum of probabilities of all sentence is one[1], in this thesis a version with a final state is presented, based on the idea of [JM09, pp. 210-226].

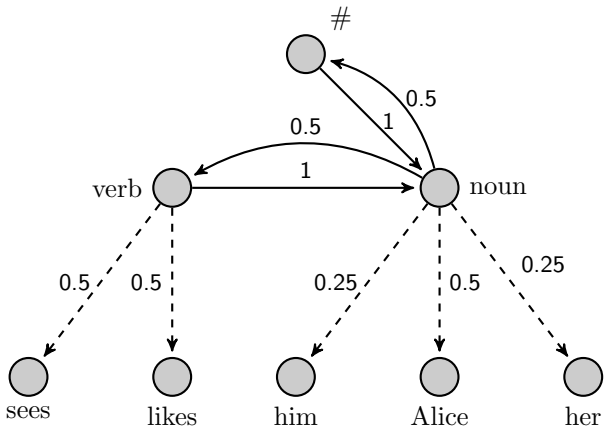
Definition 3.1: A *hidden Markov model* (HMM) is a tuple $H = (Q, V, \#, t, e)$ where

- Q is a finite, non-empty set of (hidden) states.
- V is a finite, non-empty set of observations.
- # represents the start and final state and is not a member of Q.
- t is a conditional probability distribution over $Q \cup \{\#\}$ given $Q \cup \{\#\}$, whereas $t(\# \mid \#) = 0$. t is called *transition probability* and $t(q_j \mid q_i)$ refers to the probability of transitioning from state q_i to state q_j.
- e is a conditional probability distribution over V given Q.
  e is called *emission probability* and $e(v_j \mid q_i)$ refers to the probability of observing v_j in state q_i. □

Example 3.2: Considering the corpus from the introduction,

Alice likes him Alice sees her,

the set of observations respectively words V of choice is $\{\text{Alice}, \text{likes}, \text{him}, \text{sees}, \text{her}\}$. Furthermore, one could think of the 2-element state respectively syntactic category set $Q = \{\text{verb}, \text{noun}\}$, whereas the observations *Alice*, *him* and *her* could have been emitted by the state *noun* and the observations *sees* and *likes* by the state *verb* in the sentences.
Under these assumptions using conditional relative frequency as described in the introduction to calculate t and e yields the following graphically represented HMM:



The transition probabilities are marked by sold lines, the emission probabilities are marked by dashed lines. □

A random walk in the stochastic process based on a hidden Markov model $H = (Q, V, \#, t, e)$ proceeds as follows: Beginning with the start state #, according to the conditional probability distribution t it is transitioned randomly into another state of $Q \cup \{\#\}$. Notice that because

[1] A proof for this is cannot be provided here. However, in experiments the sum of probabilities of an increasing amount of sentences tended to one.

$p(\# \mid \#) = 0$, the start state cannot transition directly into the final state.[2] In the same way, the state transitioned to randomly transitions into another one. This procedure is repeated until one transitions into the final state #. Furthermore, an observation of V is output randomly in each of these states passed except the start and final state #, according to the conditional probability distribution e.

Definition 3.3: Let $\perp \notin Q \cup \{\#\}, \perp \notin V$. For all $i \in \mathbb{N}^+$ the following random variables are defined and describe a random walk:

- $\mathbb{V}$ over V^+, where $\mathbb{V}$ represents the sequence of observations made in the random walk.
- $\mathbb{V}_i$ over $V \cup \{\perp\}$, where $\mathbb{V}_i$ represents the observation at position i of the observation sequence made in the random walk. $\mathbb{V}_i$ is assigned $\perp$ if the random walk which was created did not yield at least i observations.
- $\mathbb{Q}$ over Q^+, where $\mathbb{Q}$ represents the sequence of states passed through in the random walk, excluding the start and final state.
- $\mathbb{Q}_i$ over $Q \cup \{\perp\}$, where $\mathbb{Q}_i$ represents the state at position i of the state sequence passed through in the random walk, excluding the start and final state. $\mathbb{Q}_i$ is assigned $\perp$ if not at least i states were passed in the random walk, excluding the start and final state.
- $\mathbb{L}$ over the positive integers, where $\mathbb{L}$ represents the length of the resulting observation sequence of the random walk.

The following assumptions are made:

$$\begin{aligned} P(\mathbb{Q}_t = q_j \mid \mathbb{Q}_{t-1} = q_i) &= t(q_j \mid q_i) \text{ for all } t \geq 2, q_i, q_j \in Q \\ P(\mathbb{Q}_t = q_j \mid \mathbb{Q}_{t-1} = \perp) &= 0 \text{ for all } t \geq 2, q_j \in Q \end{aligned} \tag{1}$$

$$\begin{aligned} P(\mathbb{V}_t = v_j | \mathbb{Q}_t = q_i) &= e(v_j \mid q_i) \text{ for all } t \geq 1, v_j \in V, q_i \in Q \\ P(\mathbb{V}_t = v_j | \mathbb{Q}_t = \perp) &= 0 \text{ for all } t \geq 1, v_j \in V \end{aligned} \tag{2}$$

$$P(\mathbb{Q}_1 = q_1) = t(q_1 \mid \#) \text{ for all } q_1 \in Q \tag{3}$$

$$\begin{aligned} P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) &= t(\# \mid q_n) \text{ for all } n \geq 1, q_n \in Q \\ P(\mathbb{L} = n \mid \mathbb{Q}_n = \perp) &= 0 \text{ for all } n \geq 2 \end{aligned} \tag{4}$$

Furthermore, two important assumptions are made [Pfe13, p. 3][3]:

$$\begin{aligned} P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_{t-1} = q_{t-1}, \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}) = P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \\ \text{for all } t \geq 2, q_1, ...q_t \in Q \cup \{\perp\}, v_1, ..., v_{t-1} \in V \cup \{\perp\} \end{aligned} \tag{5}$$

$$\begin{aligned} P(\mathbb{V}_t = v_t \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_t = q_t, \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}) = P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t) \\ \text{for all } t \geq 2, q_1, ..., q_t \in Q \cup \{\perp\}, v_1, ..., v_{t-1} \in V \cup \{\perp\} \end{aligned} \tag{6}$$

Finally, the following assumptions are needed in this thesis:

$$\begin{aligned} P(\mathbb{L} = n \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n, \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n) = P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \\ \text{for all } n \geq 1, q_1, ..., q_n \in Q \cup \{\perp\}, v_1, ..., v_n \in V \cup \{\perp\} \end{aligned} \tag{7}$$

$$\begin{aligned} P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_t = q_t) = \\ P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_t) \\ \text{for all } n \geq 2, 1 \leq t \leq n-1, v_1, ..., v_n \in V \cup \{\perp\}, q_1, ..., q_t \in Q \cup \{\perp\} \end{aligned} \tag{8}$$

$$\begin{aligned} P(\mathbb{V}_t = v_t \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q} = q_1...q_n) = P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t) \\ \text{for all } n \geq 1, 1 \leq t \leq n, v_1, ..., v_t \in V \cup \{\perp\}, q_1...q_n \in Q^+ \end{aligned} \tag{9}$$

□

[2] This is no restriction since sentences to be valuated at least consist of one word.

[3] The HMM presented in [Pfe13, p. 3] do not contain a final state. Therefore the assumptions presented there are extended by allowing the random variables to have the value $\perp$.

In assumption (1) it is formalized that the probability of transitioning from a state q_i at position $t-1$ in a state sequence to state q_j at position t in a state sequence is the same for every position t in it. Likewise, the probability of observing the observation v_j at state q_i is the same for every position t at which the observation and state can be located (assumption (2)). These assumptions are called *stationarity* [Pfe13, p. 3]. However, if a random walk never reaches position t, the probability of observing an observation at t as well as transitioning into a state at $t+1$ is zero.
Analogous to this in (3) it is assumed that the probability that the first state of a state sequence is q_1 is the same as the probability of transitioning from the start state # to q_1. In the same way (4) states that the probability that q_n is the last state of a state sequence is the same as the probability of transitioning from q_n to the final state #.

The *Markov assumptions* [Pfe13, p. 3] in (5) and (6) state that given the state and observation sequence until a position t, the probability of being at $t+1$ in a specific state only depends on the state given at t. In the same way given the first $t-1$ observations and t states, the probability of emitting an specific observation at t only depends on the given state at t.

Furthermore, the following equations are used in this thesis. Equation (10) and (11) can be deduced from (5), equation (12) and (13) can be deduced from (6), equation (14), (15), (16) and (17) can be deduced from (7) and equation (18), (19) and (20) can be deduced from (8).

$$P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_{t-1} = q_{t-1}) = P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \quad \text{for all } t \geq 2, q_1, ..., q_t \in Q \cup \{\perp\} \tag{10}$$

$$P(\mathbb{Q}_t = q_t \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_{t-1} = q_{t-1}) = P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \quad \text{for all } t \geq 2, v_1, ..., v_{t-1} \in V \cup \{\perp\}, q_{t-1}, q_t \in Q \cup \{\perp\} \tag{11}$$

$$P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = q_{t-1}) = P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t) \quad \text{for all } t \geq 2, v_t \in V \cup \{\perp\}, q_{t-1}, q_t \in Q \cup \{\perp\} \tag{12}$$

$$P(\mathbb{V}_t = v_t \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = q_{t-1}) = P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t) \quad \text{for all } t \geq 2, v_1, ..., v_t \in V \cup \{\perp\}, q_{t-1}, q_t \in Q \cup \{\perp\} \tag{13}$$

$$P(\mathbb{L} = n \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n) = P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \quad \text{for all } n \geq 1, q_1, ..., q_n \in Q \cup \{\perp\} \tag{14}$$

$$P(\mathbb{L} = n, \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n) = P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \quad \text{for all } n \geq 1, v_1, ..., v_n \in V \cup \{\perp\}, q_n \in Q \cup \{\perp\} \tag{15}$$

$$P(\mathbb{L} = n, \mid \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n, \mathbb{Q}_{n-1} = q_{n-1}) = P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \quad \text{for all } n \geq 2, v_n \in V \cup \{\perp\}, q_{n-1}, q_n \in Q \cup \{\perp\} \tag{16}$$

$$P(\mathbb{L} = n, \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n, \mathbb{Q}_{n-1} = q_{n-1}) = P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \quad \text{for all } n \geq 2, v_1, ..., v_n \in V \cup \{\perp\}, q_{n-1}, q_n \in Q \cup \{\perp\} \tag{17}$$

$$P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t) = P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_t) \quad \text{for all } n \geq 2, 1 \leq t \leq n-1, v_1, ..., v_n \in V \cup \{\perp\}, q_t \in Q \cup \{\perp\} \tag{18}$$

$$P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = q_{t-1}) = P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_t) \quad \text{for all } n \geq 3, 2 \leq t \leq n-1, v_1, ..., v_n \in V \cup \{\perp\}, q_{t-1}, q_t \in Q \cup \{\perp\} \tag{19}$$

$$P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = q_{t-1}) = P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_t) \quad \text{for all } n \geq 3, 2 \leq t \leq n-1, v_t, ..., v_n \in V \cup \{\perp\}, q_{t-1}, q_t \in Q \cup \{\perp\} \tag{20}$$

Next, a pattern is shown which can be applied to deduce the equations above.

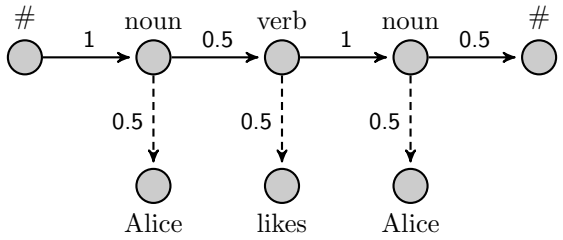
Proof: Let $\mathbb{X}, \mathbb{Y}_1, \mathbb{Y}_2, \mathbb{Z}$ be random variables over X, Y_1, Y_2, Z and let

$$P(\mathbb{X} = x \mid \mathbb{Y}_1 = y_1, \mathbb{Y}_2 = y_2, \mathbb{Z} = z) = P(\mathbb{X} = x \mid \mathbb{Z} = z) \text{ for all } x \in X, y_1 \in Y_1, y_2 \in Y_2, z \in Z. \tag{21}$$

Then for all $x \in X, y_1 \in Y_1, z \in Z$ it holds that

$$\begin{aligned}
&P(\mathbb{X} = x \mid \mathbb{Y}_1 = y_1, \mathbb{Z} = z) \\
&\quad = \frac{P(\mathbb{X} = x, \mathbb{Y}_1 = y_1, \mathbb{Z} = z)}{P(\mathbb{Y}_1 = y_1, \mathbb{Z} = z)} && \text{Definition 2.6} \\
&\quad = \frac{\sum\limits_{y_2 \in Y_2} P(\mathbb{X} = x, \mathbb{Y}_1 = y_1, \mathbb{Y}_2 = y_2, \mathbb{Z} = z)}{P(\mathbb{Y}_1 = y_1, \mathbb{Z} = z)} && \text{Theorem 2.7} \\
&\quad = \sum_{y_2 \in Y_2} \frac{P(\mathbb{X} = x, \mathbb{Y}_1 = y_1, \mathbb{Y}_2 = y_2, \mathbb{Z} = z)}{P(\mathbb{Y}_1 = y_1, \mathbb{Z} = z)} \\
&\quad = \sum_{y_2 \in Y_2} \frac{P(\mathbb{X} = x, \mathbb{Y}_1 = y_1, \mathbb{Y}_2 = y_2, \mathbb{Z} = z)}{P(\mathbb{Y}_1 = y_1, \mathbb{Y}_2 = y_2, \mathbb{Z} = z)} \cdot \frac{P(\mathbb{Y}_1 = y_1, \mathbb{Y}_2 = y_2, \mathbb{Z} = z)}{P(\mathbb{Y}_1 = y_1, \mathbb{Z} = z)} \\
&\quad = \sum_{y_2 \in Y_2} P(\mathbb{X} = x \mid \mathbb{Y}_1 = y_1, \mathbb{Y}_2 = y_2, \mathbb{Z} = z) \cdot P(\mathbb{Y}_2 = y_2 \mid \mathbb{Y}_1 = y_1, \mathbb{Z} = z) && \text{Definition 2.6} \\
&\quad = P(\mathbb{X} = x \mid \mathbb{Z} = z) \cdot \sum_{y_2 \in Y_2} P(\mathbb{Y}_2 = y_2 \mid \mathbb{Y}_1 = y_1, \mathbb{Z} = z) && (21) \\
&\quad = P(\mathbb{X} = x \mid \mathbb{Z} = z) \cdot \sum_{y_2 \in Y_2} \frac{P(\mathbb{Y}_2 = y_2, \mathbb{Y}_1 = y_1, \mathbb{Z} = z)}{P(\mathbb{Y}_1 = y_1, \mathbb{Z} = z)} && \text{Definition 2.6} \\
&\quad = P(\mathbb{X} = x \mid \mathbb{Z} = z) \cdot \frac{P(\mathbb{Y}_1 = y_1, \mathbb{Z} = z)}{P(\mathbb{Y}_1 = y_1, \mathbb{Z} = z)} && \text{Theorem 2.7} \\
&\quad = P(\mathbb{X} = x \mid \mathbb{Z} = z) && \square
\end{aligned}$$

Example 3.4: Using the HMM given in Example 3.2, an exemplarily random walk looks as follows:



Or expressed in terms of random variables: $\mathbb{Q} =$ noun verb noun, $\mathbb{V} =$ Alice likes Alice. $\square$

How can the probability of such an observation and state sequence be calculated?
In the following, a formula for $P(\mathbb{V} = v_1...v_n, \mathbb{Q} = q_1...q_n)$ is derived [JM09, p. 214] using the assumptions in Definition (3.3). Considering the definition of conditional probability (Definition 2.6) yields

$$P(\mathbb{V} = v_1...v_n, \mathbb{Q} = q_1...q_n) = P(\mathbb{V} = v_1...v_n \mid \mathbb{Q} = q_1...q_n) \cdot P(\mathbb{Q} = q_1...q_n). \tag{22}$$

Since $\mathbb{Q} = q_1...q_n \equiv \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n, \mathbb{L} = n$, for $P(\mathbb{Q} = q_1...q_n)$ from (22) using the definition of conditional probability multiple times it appears that

$$\begin{aligned}
P(\mathbb{Q} = q_1...q_n) &= P(\mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n, \mathbb{L} = n) \\
&= P(\mathbb{Q}_2 = q_2, ..., \mathbb{Q}_n = q_n, \mathbb{L} = n \mid \mathbb{Q}_1 = q_1) \cdot P(\mathbb{Q}_1 = q_1) \\
&= P(\mathbb{Q}_3 = q_3, ..., \mathbb{Q}_n = q_n, \mathbb{L} = n \mid \mathbb{Q}_1 = q_1, \mathbb{Q}_2 = q_2) \cdot \\
&\quad P(\mathbb{Q}_2 = q_2 \mid \mathbb{Q}_1 = q_1) \cdot P(\mathbb{Q}_1 = q_1) \\
&= ... \\
&= P(\mathbb{L} = n \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n) \cdot \\
&\quad \prod_{t=2}^{n} P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_{t-1} = q_{t-1}) \cdot P(\mathbb{Q}_1 = q_1).
\end{aligned}$$

With the aid of (14) and (10), this is reduced to

$$P(\mathbb{Q} = q_1...q_n) = P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \cdot \prod_{t=2}^{n} P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \cdot P(\mathbb{Q}_1 = q_1). \tag{23}$$

For $P(\mathbb{V} = v_1...v_n \mid \mathbb{Q} = q_1...q_n)$ from (22) using the definition of conditional probability in the same manner, it yields that

$$\begin{aligned}
P(\mathbb{V} = v_1...v_n \mid \mathbb{Q} = q_1...q_n) &= P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q} = q_1...q_n) = \\
&\quad P(\mathbb{L} = n \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n, \mathbb{L} = n, \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n) \cdot \\
&\quad \prod_{t=1}^{n} P(\mathbb{V}_t = v_t \mid \mathbb{Q} = q_1...q_n, \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1})
\end{aligned}$$

and with the help of (9)

$$P(\mathbb{V} = v_1...v_n \mid \mathbb{Q} = q_1...q_n) = \prod_{t=1}^{n} P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t).^4 \tag{24}$$

By plugging (23) and (24) in (22), the following is obtained:

$$\begin{aligned}
P(\mathbb{V} = v_1...v_n, \mathbb{Q} = q_1...q_n) &= P(\mathbb{Q}_1 = q_1) \cdot \prod_{t=1}^{n} P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t) \cdot \\
&\quad \prod_{t=2}^{n} P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \cdot P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \\
&= P(\mathbb{Q}_1 = q_1) \cdot P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \cdot P(\mathbb{V}_1 = v_1 \mid \mathbb{Q}_1 = q_1) \cdot \\
&\quad \prod_{t=2}^{n} P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \cdot P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t)
\end{aligned}$$

and with the help of (1), (2), (3) and (4)

$$= t(q_1 \mid \#) \cdot t(\# \mid q_n) \cdot e(v_1 \mid q_1) \cdot \prod_{i=2}^{n} t(q_i \mid q_{i-1}) \cdot e(v_i \mid q_i). \tag{25}$$

[4] Let s and s' be events. Then $P(s \mid s, s') = \frac{P(s,s,s')}{P(s,s')} = \frac{P(s,s')}{P(s,s')} = 1$. Therefore substituting s by $\mathbb{L} = n$ and s' by $\mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n, \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n$ yields that $P(\mathbb{L} = n \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_n = q_n, \mathbb{L} = n, \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n)$ equals one. Furthermore, because $P(\mathbb{L} = n \mid \mathbb{L} = n, s') = 1$ and according to the law of total probability $\sum_{m \in \mathbb{N}^+} P(\mathbb{L} = m \mid \mathbb{L} = n, s') = 1$, it holds that $P(\mathbb{L} = m \mid \mathbb{L} = n, s') = 0$ for $m \neq n$.

Therefore the probability of a state and observation sequence is calculated by multiplying the probability of starting and finishing the state sequence with the start and final state by all transition probabilities of two neighboring states in the state sequence and by the probabilities of observing the observations in their respective states.

Example 3.5: Calculating the probability of the observation and state sequence from Example 3.4 works as follows:

$$\begin{aligned} P(\mathbb{Q} = \text{noun verb noun}, \mathbb{V} = \text{Alice likes Alice}) \\ &= t(\text{noun} \mid \#) \cdot t(\text{verb} \mid \text{noun}) \cdot t(\text{noun} \mid \text{verb}) \cdot t(\# \mid \text{noun}) \cdot \\ &\quad e(\text{Alice} \mid \text{noun}) \cdot e(\text{likes} \mid \text{verb}) \cdot e(\text{Alice} \mid \text{noun}) \\ &= 1 \cdot 0.5 \cdot 1 \cdot 0.5 \cdot 0.5 \cdot 0.5 \cdot 0.5 = 0.03125. \end{aligned}$$ □

3.1 Forward and backward algorithm

However, regarding the valuation of a sentence, not the probability of a state and observation sequence, but the probability of an observation sequence respectively a sentence $v_1 ... v_n$ is of interest, since the corresponding state sequence behind it is not known.
According to the law of total probability (Theorem 2.7), $P(\mathbb{V} = v_1 ... v_n)$ can be calculated by summing over the probabilities of this observation sequence and all possible state sequences: [JM09, p. 215]

$$\begin{aligned} P(\mathbb{V} = v_1 ... v_n) &= \sum_{\substack{q_1,...,q_m \in Q, \\ m \in \mathbb{N}^+}} P(\mathbb{V} = v_1 ... v_n, \mathbb{Q} = q_1 ... q_m) \\ &= \sum_{q_1,...,q_n \in Q} P(\mathbb{V} = v_1 ... v_n, \mathbb{Q} = q_1 ... q_n) + \\ &\quad \sum_{\substack{q_1,...,q_l \in Q, \\ l \neq n, l \in \mathbb{N}^+}} P(\mathbb{V} = v_1 ... v_n, \mathbb{Q} = q_1 ... q_l). \end{aligned}$$

Considering the derivation of (24), in the cases when the lengths of the observation and state sequence differ, $P(\mathbb{L} = n \mid \mathbb{Q}_1 = q_1, ..., \mathbb{Q}_l = q_l, \mathbb{L} = l, \mathbb{V}_1 = v_1, ..., \mathbb{V}_l = v_l)$ and therefore the product which comprises this probability evaluates to zero.[4] Thus one obtains the following:

$$\begin{aligned} &= \sum_{q_1,...,q_n \in Q} P(\mathbb{Q}_1 = q_1) \cdot P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \cdot P(\mathbb{V}_1 = v_1 \mid \mathbb{Q}_1 = q_1) \cdot \\ &\qquad \prod_{t=2}^{n} P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \cdot P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t). \end{aligned} \tag{26}$$

Let $T(n, |Q|)$ denote the number of occurrences of used calculations (multiplication and addition) in (26) where n is the length of the observation and state sequence (excluding the start and final state). For $n \geq 2$ it yields that

$$T(n, |Q|) = \underbrace{(2 \cdot n)}_{\substack{\text{number of multiplications} \\ \text{in one addend}}} \cdot \underbrace{|Q|^n}_{\substack{\text{number of} \\ \text{addends}}} + \underbrace{(|Q|^n - 1)}_{\substack{\text{number of} \\ \text{additions}}}.$$

A time complexity $T(n, |Q|) \in \mathcal{O}(n \cdot |Q|^n)$ for $P(\mathbb{V} = v_1 ... v_n)$ is not feasible. To reduce the complexity, the technique of dynamic programming is useful:

> "*Hier werden zur Lösung eines Problems der Größe n alle relevanten Probleme kleinerer Größe (beginnend bei Problemgröße 1) gelöst und der Reihe nach in eine Tabelle geschrieben. Dann kann bei der Lösung eines Problems der Größe i auf die Lösung aller Probleme mit Größen $1, 2, ..., i-1$ zurückgegriffen werden, d.h. dass jedes Problem höchstens einmal gelöst wird.[...]*
> *Da bei diesem Prinzip der Aufbau der Gesamtlösung aus Teillösungen im Vordergrund steht, ordnet man ihm auch das Atrribut bottom-up zu.*" [Vog10, p. 152]

Before describing how this technique is applied, the right-hand side of (26) is analyzed from a mathematical point of view. Written in detail the equation reads as follows:

$$\begin{aligned}\sum_{q_n \in Q} \dots \sum_{q_1 \in Q} P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n)\cdot \\ & P(\mathbb{V}_n = v_n \mid \mathbb{Q}_n = q_n) \cdot \dots \cdot P(\mathbb{V}_1 = v_1 \mid \mathbb{Q}_1 = q_1)\cdot \\ & P(\mathbb{Q}_n = q_n \mid \mathbb{Q}_{n-1} = q_{n-1}) \cdot \dots \cdot P(\mathbb{Q}_2 = q_2 \mid \mathbb{Q}_1 = q_1)\cdot \\ & P(\mathbb{Q}_1 = q_1).\end{aligned}$$

To save multiplications, as much factors as possible are placed outside the brackets:

$$\begin{aligned}&\sum_{q_n \in Q} P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n)\cdot \\ &\quad P(\mathbb{V}_n = v_n \mid \mathbb{Q}_n = q_n) \cdot \sum_{q_{n-1} \in Q} P(\mathbb{Q}_n = q_n \mid \mathbb{Q}_{n-1} = q_{n-1})\cdot \\ &\quad \dots \cdot \\ &\quad P(\mathbb{V}_3 = v_3 \mid \mathbb{Q}_3 = q_3) \cdot \sum_{q_2 \in Q} P(\mathbb{Q}_3 = q_3 \mid \mathbb{Q}_2 = q_2)\cdot \\ &\quad \underbrace{P(\mathbb{V}_2 = v_2 \mid \mathbb{Q}_2 = q_2) \cdot \sum_{q_1 \in Q} P(\mathbb{Q}_2 = q_2 \mid \mathbb{Q}_1 = q_1) \cdot \underbrace{P(\mathbb{V}_1 = v_1 \mid \mathbb{Q}_1 = q_1) \cdot P(\mathbb{Q}_1 = q_1)}_{(*)}}_{(**)}.\end{aligned} \tag{27}$$

Considering the expressions marked with $(*)$ and $(**)$, one can see that

- $(*)$ is independent from all possible values which can be assigned to $q_n, ..., q_2$ and only depends on q_1.
- $(**)$ is independent from all possible values which can be assigned to $q_n, ..., q_3$ and only depends on q_2 and q_1.

To save calculations, it therefore makes sense to calculate $(*)$ only once for every value which can be assigned to q_1. $(*)$ is equal to $P(\mathbb{V}_1 = v_1, \mathbb{Q}_1 = q_1)$. Likewise, $(**)$ is calculated once for all values of q_2. In so doing, the previously calculated $P(\mathbb{V}_1 = v_1, \mathbb{Q}_1 = q_1)$ can be used.

By continuing this approach and summarizing all possible expressions in (27) as done for $(*)$ and $(**)$, one yields the *forward algorithm* [JM09, p. 217] to compute $P(\mathbb{V} = v_1...v_n)$:

Algorithm 3.1.1 Forward algorithm

Input: HMM $H = (Q, V, \#, t, e)$,
observation sequence $v_1...v_n \in V^+$
Variables: $P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t)$ for all $1 \le t \le n, q_t \in Q$,
$P(\mathbb{V} = v_1...v_n)$
Output: $P(\mathbb{V} = v_1...v_n)$

for all $q_1 \in Q$ **do**
  $P(\mathbb{V}_1 = v_1, \mathbb{Q}_1 = q_1) = e(v_1 \mid q_1) \cdot t(q_1 \mid \#)$
for all $t = 2, ..., n$ **do**
  for all $q_t \in Q$ **do**
    $P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t) = e(v_t \mid q_t) \cdot \sum_{q_{t-1} \in Q} t(q_t \mid q_{t-1})\cdot P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_{t-1} = q_{t-1})$
$P(\mathbb{V} = v_1...v_n) = \sum_{q_n \in Q} t(\# \mid q_n) \cdot P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n)$
output $P(\mathbb{V} = v_1...v_n)$

Line 2 holds as can be seen from $(*)$. Next it is shown that line 5 holds:[5]

$$
\begin{aligned}
&P(\mathbb{V}_1{=}\, v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t) \\
&\quad = \sum_{q_{t-1} \in Q} P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = q_{t-1}) + && \text{Theorem 2.7} \\
&\qquad\qquad P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = \bot)
\end{aligned}
$$

where $P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = \bot) = 0$, since it can be decomposed to $P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = \bot) \cdot P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = \bot) \cdot P(\mathbb{Q}_{t-1} = \bot)$ and $P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = \bot)$ is assigned zero according to (1).

$$
\begin{aligned}
&= \sum_{q_{t-1} \in Q} P(\mathbb{V}_t = v_t \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_t = q_t, \mathbb{Q}_{t-1} = q_{t-1}) \cdot && \text{Definition 2.6} \\
&\qquad P(\mathbb{Q}_t = q_t \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_{t-1} = q_{t-1}) \cdot \\
&\qquad P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_{t-1} = q_{t-1}) \\
&= P(\mathbb{V}_t = v_t \mid \mathbb{Q}_t = q_t) \cdot && (11), (13) \\
&\quad \sum_{q_{t-1} \in Q} P(\mathbb{Q}_t = q_t \mid \mathbb{Q}_{t-1} = q_{t-1}) \cdot \\
&\qquad P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_{t-1} = q_{t-1}) \\
&= e(v_t \mid q_t) \cdot && (2), (1) \\
&\quad \sum_{q_{t-1} \in Q} t(q_t \mid q_{t-1}) \cdot P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_{t-1} = v_{t-1}, \mathbb{Q}_{t-1} = q_{t-1})
\end{aligned}
$$

In the same way line 6 can be shown:

$$
\begin{aligned}
P(\mathbb{V} = v_1 ... v_n) &= \sum_{q_n \in Q} P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{L} = n, \mathbb{Q}_n = q_n) + && \text{Theorem 2.7} \\
&\qquad P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{L} = n, \mathbb{Q}_n = \bot) \\
&= \sum_{q_n \in Q} P(\mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n) \cdot && \text{Definition 2.6,} \\
&\qquad P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n) && (15), (4) \\
&= \sum_{q_n \in Q} P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \cdot && (15) \\
&\qquad P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n) \\
&= \sum_{q_n \in Q} t(\# \mid q_n) \cdot P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n) && (4)
\end{aligned}
$$

This approach used in the forward algorithm corresponds to the technique of dynamic programming mentioned before:
The solution to a problem of size (t, q_t) - the probability of the observation sequence $v_1 ... v_t$ and the state q_t at position t - is calculated by summing over the solutions to problems of size $(t-1, q_{t-1})$, $q_{t-1} \in Q$, whereas each suchlike solution to a problem is multiplied both by the probability of transitioning from state q_{t-1} to state q_t and by the probability of observing the observation v_t at position t at state q_t.
The probability of the observation sequence $v_1 ... v_n$ actually searched for is then calculated by using the solutions to problems of size $(n, q_n), q_n \in Q$. This means that the already calculated probabilities of observation sequences $v_1 ... v_n$, where at position n the state $q_n \in Q$ is, are multiplied by the probability of transitioning from state q_n to the final state # and then are added up.
By calculating and saving the solutions to problems of size (t, q_t) in a *bottom-up*-fashion - meaning that the probabilities of the observation sequences $v_1 ... v_t$ and a state $q_t \in Q$ at position t are

[5] Compare [Pfe13, p. 6].

calculated according to the length of sequences, from 1 to n - a problem of size (j, q_j) can be solved with the help of solutions to problems of size $(j-1, q_{j-1})$,$q_{j-1} \in Q$. Therefore the solution to a problem does not have to be calculated several times.

Next, the time complexity of the forward algorithm is analyzed. The chart below lists the number of calculations used in the forward algorithm:

	multiplications	additions	calculations	
$P(\mathbb{V}_1 = v_1, \mathbb{Q}_1 = q_1)$	1	0	$\lvert Q\rvert$	
$P(\mathbb{V}_1 = v_1, \mathbb{V}_2 = v_2, \mathbb{Q}_2 = q_2)$	$\lvert Q\rvert + 1$	$\lvert Q\rvert - 1$	$\lvert Q\rvert$	$\Big\}n-1$
...	...	...	...	
$P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n)$	$\lvert Q\rvert + 1$	$\lvert Q\rvert - 1$	$\lvert Q\rvert$	
$P(\mathbb{V} = v_1...v_n)$	$\lvert Q\rvert$	$\lvert Q\rvert - 1$	1	

Hence, the number of calculations used arises in this case as

$$T(n, |Q|) = (n-1) \cdot [(|Q| + 1 + |Q| - 1) \cdot |Q|] + 3 \cdot |Q| - 1.$$

Compared to the time complexity $\mathcal{O}(n \cdot |Q|^n)$ of the naive approach in (26), this corresponds to a time complexity of merely $T(n, |Q|) \in \mathcal{O}(n \cdot |Q|^2)$. [JM09, p. 215]

The forward algorithm bears its name, because $P(\mathbb{V} = v_1...v_n)$ is calculated by at first considering only the first observation v_1 (with all possible states at position 1), then moving forward and considering the the first two observations (with all possible states at position 2) and so on until the whole observation sequence $v_1...v_n$ is considered.
There is also another approach of calculating $P(\mathbb{V} = v_1...v_n)$: The *backward algorithm* [JM09, p. 222] at first considers the last observation v_n (for all possible states given at $n-1$), then the last two observations $v_{n-1}v_n$ (for all possible states given at $n-2$) and so on until the whole observation sequence $v_1...v_n$ is considered. The algorithm reads as follows:

Algorithm 3.1.2 Backward algorithm

Input: HMM $H = (Q, V, \#, t, e)$,
observation sequence $v_1...v_n \in V^+$
Variables: $P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_t)$ for all $1 \leq t \leq n-1, q_t \in Q$,
$P(\mathbb{V} = v_1...v_n)$
Output: $P(\mathbb{V} = v_1...v_n)$

1: **for all** $q_n \in Q$ **do**
2: $\quad P(\mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_{n-1} = q_{n-1}) = \sum_{q_n \in Q} t(q_n \mid q_{n-1}) \cdot e(v_n \mid q_n) \cdot t(\# \mid q_n)$
3: **for all** $t = n-2, ..., 1$ **do**
4: $\quad$ **for all** $q_t \in Q$ **do**
5: $\quad\quad P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_t) = \sum_{q_{t+1} \in Q} e(v_{t+1} \mid q_{t+1}) \cdot t(q_{t+1} \mid q_t) \cdot$
$\quad\quad\quad P(\mathbb{V}_{t+2} = v_{t+2}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_{t+1} = q_{t+1})$
6: $P(\mathbb{V} = v_1...v_n) = \sum_{q_1 \in Q} e(v_1 \mid q_1) \cdot t(q_1 \mid \#) \cdot$
$\quad\quad P(\mathbb{V}_2 = v_2, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_1 = q_1)$
7: output $P(\mathbb{V} = v_1...v_n)$

Next it is shown that line 2 holds:

$$\begin{aligned} &P(\mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_{n-1} = q_{n-1}) \\ &\quad = \frac{P(\mathbb{V}_n = v_n, \mathbb{L} = n, \mathbb{Q}_{n-1} = q_{n-1})}{P(\mathbb{Q}_{n-1} = q_{n-1})} \qquad \text{Definition 2.6} \end{aligned}$$

$$
\begin{aligned}
&= \frac{\sum_{q_n \in Q \cup \{\bot\}} P(\mathbb{V}_n = v_n, \mathbb{L} = n, \mathbb{Q}_n = q_n, \mathbb{Q}_{n-1} = q_{n-1})}{P(\mathbb{Q}_{n-1} = q_{n-1})} && \text{Theorem 2.7}\\
&= \sum_{q_n \in Q \cup \{\bot\}} P(\mathbb{V}_n = v_n, \mathbb{L} = n, \mathbb{Q}_n = q_n \mid \mathbb{Q}_{n-1} = q_{n-1}) && \text{Definition 2.6}\\
&= \sum_{q_n \in Q \cup \{\bot\}} P(\mathbb{L} = n \mid \mathbb{V}_n = v_n, \mathbb{Q}_n = q_n, \mathbb{Q}_{n-1} = q_{n-1}) \cdot && \text{Definition 2.6}\\
&\qquad P(\mathbb{V}_n = v_n \mid \mathbb{Q}_n = q_n, \mathbb{Q}_{n-1} = q_{n-1}) \cdot\\
&\qquad P(\mathbb{Q}_n = q_n \mid \mathbb{Q}_{n-1} = q_{n-1})\\
&= \sum_{q_n \in Q \cup \{\bot\}} P(\mathbb{L} = n \mid \mathbb{Q}_n = q_n) \cdot && (16), (12)\\
&\qquad P(\mathbb{V}_n = v_n \mid \mathbb{Q}_n = q_n) \cdot\\
&\qquad P(\mathbb{Q}_n = q_n \mid \mathbb{Q}_{n-1} = q_{n-1})\\
&= \sum_{q_n \in Q} t(\# \mid q_n) \cdot e(v_n \mid q_n) \cdot t(q_n \mid q_{n-1}) && (1), (2), (4)
\end{aligned}
$$

Line 5 holds as well:[6]

$$
\begin{aligned}
&P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_t)\\
&\quad = \sum_{q_{t+1} \in Q \cup \{\bot\}} P(\mathbb{V}_{t+2} = v_{t+2}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n, \mathbb{V}_{t+1} = v_{t+1}, \mathbb{Q}_{t+1} = q_{t+1} \mid \mathbb{Q}_t = q_t) && \text{Theorem 2.7}\\
&\quad = \sum_{q_{t+1} \in Q \cup \{\bot\}} P(\mathbb{V}_{t+2} = v_{t+2}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_{t+1} = v_{t+1}, \mathbb{Q}_{t+1} = q_{t+1}, \mathbb{Q}_t = q_t) \cdot && \text{Definition 2.6}\\
&\qquad P(\mathbb{V}_{t+1} = v_{t+1} \mid \mathbb{Q}_{t+1} = q_{t+1}, \mathbb{Q}_t = q_t) \cdot\\
&\qquad P(\mathbb{Q}_{t+1} = q_{t+1} \mid \mathbb{Q}_t = q_t)\\
&\quad = \sum_{q_{t+1} \in Q \cup \{\bot\}} P(\mathbb{V}_{t+1} = v_{t+1} \mid \mathbb{Q}_{t+1} = q_{t+1}) \cdot && (12), (20)\\
&\qquad P(\mathbb{Q}_{t+1} = q_{t+1} \mid \mathbb{Q}_t = q_t) \cdot\\
&\qquad P(\mathbb{V}_{t+2} = v_{t+2}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_{t+1} = q_{t+1})\\
&\quad = \sum_{q_{t+1} \in Q} e(v_{t+1} \mid q_{t+1}) \cdot t(q_{t+1} \mid q_t) \cdot P(\mathbb{V}_{t+2} = v_{t+2}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_{t+1} = q_{t+1}) && (1), (2), (4)
\end{aligned}
$$

And so does line 6:

$$
\begin{aligned}
&P(\mathbb{V} = v_1 ... v_n) = P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{L} = n)\\
&\quad = \sum_{q_1 \in Q \cup \{\bot\}} P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{L} = n, \mathbb{Q}_1 = q_1) && \text{Theorem 2.7}\\
&\quad = \sum_{q_1 \in Q \cup \{\bot\}} P(\mathbb{V}_2 = v_2, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_1 = v_1, \mathbb{Q}_1 = q_1) \cdot && \text{Definition 2.6}\\
&\qquad P(\mathbb{V}_1 = v_1 \mid \mathbb{Q}_1 = q_1) \cdot\\
&\qquad P(\mathbb{Q}_1 = q_1)\\
&\quad = \sum_{q_1 \in Q \cup \{\bot\}} P(\mathbb{V}_2 = v_2, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_1 = q_1) \cdot && (18)\\
&\qquad P(\mathbb{V}_1 = v_1 \mid \mathbb{Q}_1 = q_1) \cdot\\
&\qquad P(\mathbb{Q}_1 = q_1)\\
&\quad = \sum_{q_1 \in Q} P(\mathbb{V}_2 = v_2, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_1 = q_1) \cdot e(v_1 \mid q_1) \cdot t(q_1 \mid \#) && (1), (2), (4)
\end{aligned}
$$

[6] Compare [Pfe13, p. 7].

Just as the forward algorithm, the backward algorithm uses the technique of dynamic programming and has a time complexity of $\mathcal{O}(n \cdot |Q|^2)$.
Using these more efficient methods of calculation, the probability model of the hidden Markov model on V^+ reads as follows:

$$\begin{aligned} \mathcal{M}_{HMM} = \{p_H : V^+ \to [0,1] \mid H = (Q, V, \#, t, e) \text{ hidden Markov model}\} \\ p_H(v_1...v_n) = P(\mathbb{V} = v_1...v_n) \end{aligned} \tag{28}$$

Either the forward algorithm or the backward algorithm can be used to calculate $P(\mathbb{V} = v_1...v_n)$.

3.2 Baum-Welch algorithm

Up to this point, a corpus was regarded as a collection of sentences. Now, a corpus will be considered as a function $h : X \to \mathbb{R}_{\geq 0}$ as described in Definition 2.8, where X is the set of all possible sentences and h assigns each sentence the number of occurrences in the collection of sentences. Because each sentence is processed individually without considering its position in the collection of sentences, these two notions of a corpus are equivalent.

In the earlier section the hidden Markov model and the procedure of calculating the probability of an observation sequence given an HMM were introduced. An essential missing aspect is how to determine suitable parameters of an HHM for a given corpus. Obtaining the set of observations is done straightforwardly by considering all words appearing in the corpus. Then the way a HMM $H = (Q, V, \#, t, e)$ fits to a corpus $h : V^+ \to \mathbb{R}_{\geq 0}$ is expressed by $L(h, p_H)$, the likelihood of h under p_H, as described in Definition 2.10. Therefore, the maximum likelihood estimate $mle(h, \mathcal{M}_{HMM})$, the probability distribution p_H of the probability model $\mathcal{M}_{HMM}$ given in 28 which maximizes $L(h)$, has to be found. Because p_H is induced by the HMM $H = (Q, V, \#, t, e)$, finding a maximizing p_H means finding the maximizing parameters Q, t and e.
Assuming complete data, which means that for each sentence in the corpus the state sequence which generated the sentence is known, Q can be be extracted from the state sequences in the corpus and t and e can be calculated directly with the help of conditional relative frequency [MS99, pp. 345-348] [Pfe13, p. 14].
However, if only incomplete data is present, which means that merely the sentences of the corpus are known, Q, t and e cannot be calculated directly. Regarding the states, not their concrete names, but the number of states is important, as the states are not used nominally. Determining a suitable number of states requires advanced procedures, but given a number of states, the *Baum-Welch algorithm* [BPSW70] which is an instance of the *Expectation-Maximization (EM) algorithm* [DLR77] [JM09, p. 221] can be used to iteratively specify the conditional probability distributions t and e. The Baum-Welch algorithm takes as an input an initial HMM $H = (Q, V, \#, t_0, e_0)$ and a corpus $h : V^+ \to \mathbb{R}_{\geq 0}$ and calculates a sequence of pairs of tuples of conditional probability distributions $(t_1, e_1), (t_2, e_2), ...$ Because the Baum-Welch algorithm is an instance of the EM algorithm, it holds that $L(h, p_{(Q,V,\#,t_k,e_k)}) \leq L(h, p_{(Q,V,\#,t_{k+1},e_{k+1})})$ for all $k \in \mathbb{N}$ [DLR77]. The sequence converges to a tuple of conditional probability distributions which at least reaches a saddle point or locally maximizes the likelihood [Vog12] of the corpus given V and Q.

The Baum-Welch algorithm reads as follows: [JM09, p. 226]

Algorithm 3.2.1 Baum-Welch algorithm

Input: HMM $H = (Q, V, \#, t_0, e_0)$,
corpus $h : V^+ \to \mathbb{R}_{\geq 0}$

Variables: $t : Q \cup \{\#\} \to \mathcal{M}(Q \cup \{\#\})$, $e : Q \to \mathcal{M}(V)$,
$t_l : Q \cup \{\#\} \to \mathcal{M}(Q \cup \{\#\})$, $e_l : Q \to \mathcal{M}(V)$, for all $l \geq 1$,
$count_{trans} : (Q \cup \{\#\}) \times (Q \cup \{\#\}) \to \mathbb{R}_{\geq 0}$, $count_{em} : V \times Q \to \mathbb{R}_{\geq 0}$.

Output: A sequence of tuples of conditional probability distributions $(t_1, e_1), (t_2, e_2), ...$

1: **for all** $k = 1, 2, ...$ **do**
2: $(t, e) := (t_{k-1}, e_{k-1})$
3: $count_{trans}(q_j, q_i) := 0$ for every $q_j, q_i \in Q \cup \{\#\}$
4: $count_{em}(v_j, q_i) := 0$ for every $v_j \in V, q_i \in Q$
5: **for all** $v = v_1...v_n \in supp(h)$ **do**
6: **for all** $t = 1, 2, ...n - 1$ **do**
7: **for all** $q_j, q_i \in Q$ **do**
8: $count_{trans}(q_j, q_i) = count_{trans}(q_j, q_i) + h(v) \cdot P(\mathbb{Q}_t = q_i, \mathbb{Q}_{t+1} = q_j \mid \mathbb{V} = v_1...v_n)$
9: **for all** $t = 1, 2, ...n$ **do**
10: **for all** $q_i \in Q$ **do**
11: $count_{em}(v_t, q_i) = count_{em}(v_t, q_i) + h(v) \cdot P(\mathbb{Q}_t = q_i \mid \mathbb{V} = v_1...v_n)$
12: **for all** $q_i \in Q$ **do**
13: $count_{trans}(q_i, \#) = count_{trans}(q_i, \#) + h(v) \cdot P(\mathbb{Q}_1 = q_i \mid \mathbb{V} = v_1...v_n)$
14: $count_{trans}(\#, q_i) = count_{trans}(\#, q_i) + h(v) \cdot P(\mathbb{Q}_n = q_i \mid \mathbb{V} = v_1...v_n)$
15: **for all** $q_j, q_i \in Q \cup \{\#\}$ **do**
16: $t_k(q_j \mid q_i) = \frac{count_{trans}(q_j, q_i)}{\sum_{q_{j'} \in Q} count_{trans}(q_{j'}, q_i)}$
17: **for all** $v_i \in V, q_i \in Q$ **do**
18: $e_k(v_i \mid q_i) = \frac{count_{em}(v_i, q_i)}{\sum_{v_{i'} \in V} count_{em}(v_{i'}, q_i)}$
19: output (t_k, e_k)

Starting with a pair of initial conditional probability distributions (t_0, e_0), in line two the pair (t, e) which it is worked with in the current iteration is set to the pair calculated in the last iteration. In line three and four $count_{trans}$ and $count_{em}$ are initialized. $count_{trans}(q_j, q_i)$ represents the relevance for state q_i to transition into state q_j in possible state sequences for all observation sequences in the corpus, given the current (t, e). In the same way $count_{em}(v_j, q_i)$ represents the relevance for an observation v_j being emitted in state q_i. Both counts are calculated using the current pair of conditional probability distributions (t, e) and the corpus h in the expectation step, which is comprised in lines five to 14. Once these counts are calculated, they are used in lines 15 to 18 in the maximization step to calculate the next pair of conditional probability distributions.
Let $q_j, q_i \in Q$. Then $count_{trans}(q_j, q_i)$ is calculated in the expectation step by summing over all observation sequences $v_1...v_n$ of the support of the corpus and positions t in these sequences, adding the probability of having the corresponding state sequence contain q_i at position t and q_j at position $t + 1$, given $v_1...v_n$, and multiplying this by the frequency of $v_1...v_n$ according to the corpus. Because q_j and q_i are elements of Q, positions until $n - 1$ are considered. Different from [JM09, p. 226] where one observation sequence is considered, here it is iterated over different observation sequences in the corpus as done in other instances of the Baum-Welch algorithm as well [Vog12]. Another difference is that in line 13 and 14 for a given observation sequence the relevance for a state $q_i \in Q$ to transition into the final state # respectively the relevance the start state # to transition into state $q_i \in Q$ is calculated. This is done in a similar way by calculating the probability that given the observation sequence, q_i is the first respectively the last state of the corresponding state sequence. Note that the result of calculating this probability depends on the current (t, e).
Iterating over all observation sequences of the corpus, $count_{em}$ is calculated in line nine to eleven

by calculating for each position t of a given observation sequence and for each $q_i \in Q$ the probability that q_i emitted the observation at position t of this sequence.

Next it is shown how to calculate $P(\mathbb{Q}_t = q_i, \mathbb{Q}_{t+1} = q_j \mid \mathbb{V} = v_1...v_n)$ and $P(\mathbb{Q}_t = q_j \mid \mathbb{V} = v_1...v_n)$ from the Baum-Welch algorithm above [JM09, pp. 223-225].
All of these expressions can be transformed by using the definition of conditional probability:

$$P(\mathbb{Q}_t = q_i, \mathbb{Q}_{t+1} = q_j \mid \mathbb{V} = v_1...v_n) = \frac{P(\mathbb{Q}_t = q_i, \mathbb{Q}_{t+1} = q_j, \mathbb{V} = v_1...v_n)}{P(\mathbb{V} = v_1...v_n)} \tag{29}$$

$$P(\mathbb{Q}_t = q_j \mid \mathbb{V} = v_1...v_n) = \frac{P(\mathbb{Q}_t = q_j, \mathbb{V} = v_1...v_n)}{P(\mathbb{V} = v_1...v_n)} \tag{30}$$

$P(\mathbb{V} = v_1...v_n)$ can be calculated by using the forward or backward algorithm.
In the following the numerators are calculated by decomposing them into probabilities, which can either be gained from (1), (2), (4) or be calculated alongside performing the forward and backward algorithm.
For $1 \leq t < n-1$ and $t = n-1$, respectively, $P(\mathbb{Q}_t = q_i, \mathbb{Q}_{t+1} = q_j, \mathbb{V} = v_1...v_n)$ from (29) can be decomposed to

$$\begin{aligned}
&P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_i)\cdot \\
&P(\mathbb{Q}_{t+1} = q_j \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_i)\cdot \\
&P(\mathbb{V}_{t+1} = v_{t+1} \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_i, \mathbb{Q}_{t+1} = q_j)\cdot \\
&P(\mathbb{V}_{t+2} = v_{t+2}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{t+1} = v_{t+1}, \mathbb{Q}_t = q_i, \mathbb{Q}_{t+1} = q_j)
\end{aligned}$$

and

$$\begin{aligned}
&P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_{n-1} = v_{n-1}, \mathbb{Q}_{n-1} = q_i)\cdot \\
&P(\mathbb{Q}_n = q_j \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{n-1} = v_{n-1}, \mathbb{Q}_{n-1} = q_i)\cdot \\
&P(\mathbb{V}_n = v_n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_{n-1} = v_{n-1}, \mathbb{Q}_{n-1} = q_i, \mathbb{Q}_n = q_j)\cdot \\
&P(\mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_{n-1} = q_i, \mathbb{Q}_n = q_j),
\end{aligned}$$

respectively, by applying the definition of conditional probability multiple times. Using equations (11), (13), (19) and (17) this reduces to

$$\begin{aligned}
&P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_i)\cdot \\
&P(\mathbb{Q}_{t+1} = q_j \mid \mathbb{Q}_t = q_i)\cdot \\
&P(\mathbb{V}_{t+1} = v_{t+1} \mid \mathbb{Q}_{t+1} = q_j)\cdot \\
&P(\mathbb{V}_{t+2} = v_{t+2}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_{t+1} = q_j)
\end{aligned}$$

and

$$\begin{aligned}
&P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_{n-1} = v_{n-1}, \mathbb{Q}_{n-1} = q_i)\cdot \\
&P(\mathbb{Q}_n = q_j \mid \mathbb{Q}_{n-1} = q_i)\cdot \\
&P(\mathbb{V}_n = v_n \mid \mathbb{Q}_n = q_j)\cdot \\
&P(\mathbb{L} = n \mid \mathbb{Q}_n = q_j),
\end{aligned}$$

respectively. Next, consider $P(\mathbb{Q}_t = q_j, \mathbb{V} = v_1...v_n)$ from (30). For $1 \leq t < n$ and $t = n$, respectively, it can be decomposed to

$$\begin{aligned}
&P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_j)\cdot \\
&P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_j),
\end{aligned}$$

and

$$\begin{aligned}
&P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_j)\cdot \\
&P(\mathbb{L} = n \mid \mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_j),
\end{aligned}$$

respectively, in the same way. By applying definition (18) and (15) this is reduced to

$$P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_j) \cdot$$
$$P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_j),$$

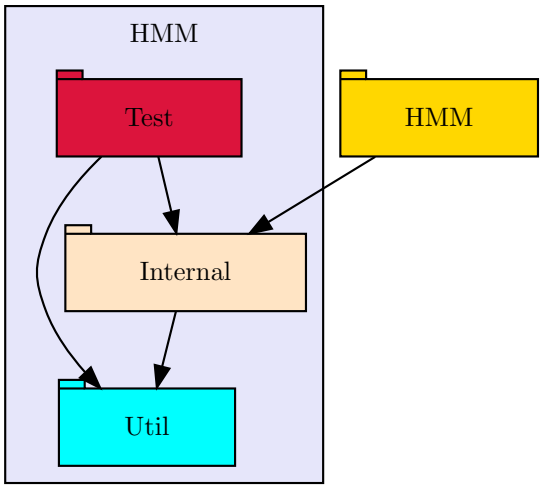
and

$$P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_n = v_n, \mathbb{Q}_n = q_j) \cdot$$
$$P(\mathbb{L} = n \mid \mathbb{Q}_n = q_j),$$

respectively. Finally, all components to perform the Baum-Welch algorithm are known.

4 Implementation of the Baum-Welch algorithm

The program which implements hidden Markov models as described in section 3 is written in Haskell. In the following, the structure of the program and some implementations of functions are described exemplarily.
The following diagram created with the Haskell package Source Graph [Mil13] shows the modules the program consists of:



The *HMM.Internal* module comprises all functions related to HMM. Since not every function is useful for the end user, the *HMM* module imports every function from *HMM.Internal* and exports only these necessary for the end user. The *HMM.Test* module contains functions which help to ensure that HMM are implemented correctly. Both the *HMM.Test* and *HMM.Internal* module use the *HMM.Util* module which consists of auxiliary functions which are not related to HMM.

One problem occurring when implementing HMM is the problem of *underflowing*. Consider the following example of a summation over the product of small probabilities. This could occur in a similar way in the naive approach of calculating the probability of an observation sequence.

```
Prelude> sum $ take 10 $ repeat (1e-162 * 1e-162)
0.0
```

```
Prelude> 1e-323
1e-323
```

```
Prelude> 1e-324
0.0
```

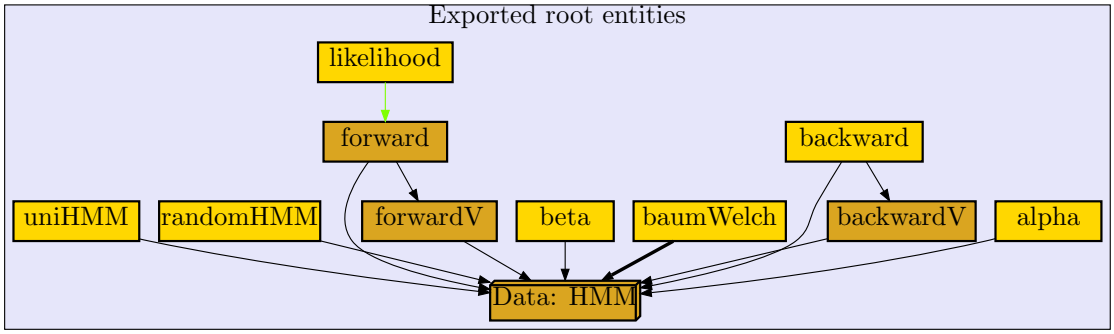
Although $10 \cdot 10^{-162} \cdot 10^{-162} = 10^{-323}$, which could be stored in the computer memory, the computation evaluates to zero because $10^{-162} \cdot 10^{-162} = 10^{-324}$ underflows. This is especially a problem in the field of bioinformatics [MS99, p. 340] when long gene sequences are considered [Man06, p. 1]. One approach with the aim to prevent underflowing is to calculate in the *logarithmic domain* as described in [Man06]. The Haskell module *Data.Number.LogFloat* [wnt10] implements this approach. The only modification which has to be done is to use the logarithm of the probabilities which is achieved by the *logFloat* function. Then the *LogFloat* instance of the *Num* class takes care of performing addition and multiplication in the logarithmic domain correctly:

```
Prelude> sum $ take 10
             $ repeat ( logFloat (1e-162 :: Double) *
                        logFloat (1e-162 :: Double) )
LogFloat 1.0e-323
```

As the source code of Data.Number.LogFloat states, calculating in the logarithmic domain is a bit slower and less accurate in the last decimal places, however it helps to prevent underflow.

4.1 The HMM.Internal module

The following diagram, also created with the Haskell package Source Graph, shows the data type, the functions and their dependencies inside the *HMM.Internal* module:



uniHMM and *randomHMM* are functions that can be used to create an HMM. *forward*, *backward* and *baumWelch* are implementations of the corresponding algorithms in section 3. *likelihood* uses the *forward* function and can be used to calculate the likelihood for a given HMM and corpus. *forwardV*, *alpha*, *beta* and *backwardV* are auxiliary functions which are also used by *HMM.Test*.

The export and import section of *HMM.Internal* looks as follows:[7]

```
1  module HMM.Internal
2    ( -- *  HMM
3      HMM(..)
4      -- *  Construction
5      -- ** Initialization
6    , uniHMM, randomHMM
7      -- * Algorithms
8    , forward, backward, baumWelch
9    , forwardV, backwardV
10   , alpha, beta
11     -- * Likelihood
12   , likelihood
13   ) where
14
15 import Control.Monad ( foldM, mapM )
16 import Data.Hashable ( Hashable )
17 import qualified Data.HashMap.Lazy as M ( HashMap, (!), fromList, member )
18 import qualified Data.List as L ( foldl' )
19 import qualified Data.MemoCombinators as Memo ( integral, memo2 )
20 import Data.Number.LogFloat ( LogFloat, logFloat )
21 import qualified Data.Vector.Unboxed as V ( Vector, generate, imap, ifoldr'
22                                          , fromList, length, replicate
23                                          , unsafeAccum, unsafeIndex, )
24 import HMM.Util ( nubOrd, calcIndex, randomDist, sumIndices )
```

4.1.1 Data structure for HMM

```
1  type Dist = V.Vector LogFloat
2
3  data HMM state observation = HMM
4    { numberOfStates       :: Int
5    , numberOfObservations :: Int
6    , transitionDist       :: Dist
7    , emissionDist         :: Dist
8    , stateMap             :: M.HashMap state Int
9    , observationMap       :: M.HashMap observation Int
10   }
```

[7] Due to lack of space and for reasons of clarity, in the following comments in the source code are omitted.

An HMM is realized as an algebraic data type with one value constructor which contains six fields: The first two fields comprise values of type *Int* and represent the number of states (excluding the start and final state) and the number of observations of the HMM.
In the fifth and sixth field the states and observations of an HMM are stored, together with a mapping to unique Integers, starting from zero to the number of states and observations minus one, respectively.
The third and fourth field represent the two conditional probability distributions of an HMM, whereas probabilities are stored in an unboxed vector, designed especially for element types [Les12]. Note that if the HMM contains m states, represented by Integers from 0 to $m-1$, the Integer m represents the start and final state # and the vector for transition probabilities has a size of $(m+1) \cdot (m+1)$. To receive the probability for an observation/state given a state, the respective mappings are used to yield the corresponding Integers. These can be used together with the *calcIndex* function from *HMM.Util* to calculate the index at which the requested probability is stored.
Note that although *HMM state observation* is polymorphic in the *state* and *observation* type, functions which use HMM require *state* and *observation* for example to be in the *Hashable* typeclass, because a hash map is used for the mapping to Integers.

4.1.2 Creating HMM

As mentioned before, two functions are provided to create HMM:

- *uniHMM*
- *randomHMM*

The type signature of *uniHMM* reads as follows:

```
1 uniHMM :: (Ord state, Ord observation, Hashable state, Hashable observation)
2        => [state]
3        -> [observation]
4        -> Maybe (HMM state observation)
```

uniHMM takes a list of states and a list of observations as an input. Duplicate states and observations are removed before proceeding. Because the set based *nubOrd* function from *HMM.Util* is used for this, the *state* and *observation* type need to be in the *Ord* typeclass.
If zero states or zero observations are given as an input, *Nothing* is returned. Otherwise *uniHMM* returns an HMM where for all states the probability of observing an observation is distributed uniformly. The same holds for the conditional probability distribution t, except that the probability of transitioning from # to # is zero as described in Definition 3.1.

randomHMM acts as *uniHMM* but provides randomly distributed probabilities higher than zero.

Section 5 shows exemplarily that *randomHMM* is preferred over *uniHMM* and leads to a higher likelihood. [MS99, p. 339] states that random initial transition probabilities are "normally satisfactory", however appropriate emission probabilities "turn out to be particularly important" in order to find a global maximum of the likelihood function for a given corpus. The emission probabilities could tried to be "roughly estimated".
An approach I tried was to determine emission probabilities by using relative frequency of the observations in the corpus and changing the emission probabilities for each state slightly. However, this did not yield better results.
[Rab89, p. 274] describes further approaches for setting initial emission probabilities.

4.1.3 Forward algorithm

```
1  forward :: (Ord state, Ord observation, Hashable state, Hashable observation)
2          => HMM state observation
3          -> [observation]
4          -> Maybe LogFloat
5  forward hmm@(HMM _ _ _ _ _ observationMap) os =
6    if length os == 0 ||
7       not (all (`M.member` observationMap) os)
8      then Nothing
9      else let osV = V.fromList . map (\o -> observationMap M.! o) $ os
10          in  Just $ forwardV hmm osV
```

forward takes an HMM and an observation sequence as an input. If no observations are present or if the observations in the sequence are not part of the observation set of the HMM, *Nothing* is returned. Otherwise, the observations are transformed into Integers and *forwardV* is evoked.

```
11 forwardV :: (Ord state, Ord observation, Hashable state, Hashable observation)
12         => HMM state observation -> V.Vector Int -> LogFloat
13 forwardV (HMM m n tDist eDist _ _) osV =
14   let l    = V.length osV
15   in  L.foldl' (\prob state ->
16                 let a         = alpha (l-1) state
17                     transProb = tDist `V.unsafeIndex`
18                                 calcIndex state (m+1) m
19                 in  prob + transProb * a) (logFloat (0 :: Double)) [0..(m-1)]
20     where alpha = Memo.memo2 Memo.integral Memo.integral alpha'
21           alpha' 0 state =
22             let emissionProb   = eDist `V.unsafeIndex`
23                                  calcIndex state n (osV `V.unsafeIndex` 0)
24                 transitionProb = tDist `V.unsafeIndex`
25                                  calcIndex m (m+1) state
26             in  emissionProb * transitionProb
27           alpha' t state =
28             let emissionProb = eDist `V.unsafeIndex`
29                                calcIndex state n (osV `V.unsafeIndex` t)
30                 as = L.foldl' (\prob preState ->
31                                  let a         = alpha (t-1) preState
32                                      transProb = tDist `V.unsafeIndex`
33                                                  calcIndex preState (m+1) state
34                                  in  prob + transProb * a)
35                               (logFloat (0 :: Double)) [0..(m-1)]
36             in  emissionProb * as
```

forwardV is a realization of the forward algorithm described in Section 3.1. Line 15 to line 19 in the implementation correspond to the termination in line 6 of the forward algorithm. Line 21 to line 26 refer to the initialization in line 2 and line 27 to line 36 refer to line 5.
alpha t state is used as a paraphrase, because it is common to abbreviate $P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = state)$ by $\alpha_t(state)$ [JM09, p. 215]. [8]
The technique of dynamic programming is implemented by using the *memo2* and *integral* functions from *Data.MemoCombinators* in line 20. These functions turn *alpha* into a function which memorizes passed arguments and corresponding results. The memorizing table is not discarded until the *forwardV* function is left.

[8] In the same way *beta t state* used in the *beta* function corresponds to $\beta_t(state)$ which is an abbreviation for $P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V}_n = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = state)$ (compare [JM09, p. 222]).

4.1.4 Baum-Welch algorithm

```
1  baumWelch :: (Ord state, Ord observation,
2                Hashable state, Hashable observation)
3            => HMM state observation
4            -> [[observation]]
5            -> Int
6            -> Maybe (HMM state observation)
7  baumWelch hmm@(HMM _ _ _ _ _ observationMap) corpus i =
8    if corpus == [[]] || corpus == [] || not (all (not . null) corpus) ||
9       not (all (all (`M.member` observationMap)) corpus)
10     then Nothing
11     else let corpusV = map (V.fromList . map (\o -> observationMap M.! o)) corpus
12          in  Just $ baumWelch' hmm corpusV i
13   where baumWelch' hmm corpusV 0 = hmm
14         baumWelch' hmm corpusV n = let newHmm = mstep $ estep hmm corpusV
15                                    in  baumWelch' newHmm corpusV (n-1)
```

The *baumWelch* function takes an HMM, a list of observation sequences and the number of iterations the Baum-Welch algorithm is to perform as an input. Similar to the *forward* function, at first the observations in the corpus are verified to be comprised by the HMM. If they do, the observations are transformed to Integers and *estep* and *mstep* of the Baum-Welch algorithm are performed alternating until the requested number of iterations is reached.

```
16         estep hmm@(HMM m n tDist eDist stateMap observationMap) corpusV =
17           let tDist_initCount = V.fromList $ replicate ((m+1) * (m+1))
18                                               (logFloat (0 :: Double))
19               eDist_initCount = V.fromList $ replicate (m * n)
20                                               (logFloat (0 :: Double))
21           in  L.foldl' (\(HMM _ _ !tDist_count !eDist_count _ _) osV ->
22                 let (tDistMid, eDistMid, tDistInit, tDistFinal)
23                         = calcPartCounts hmm osV
24
25                     tDist_count' = V.unsafeAccum (+) tDist_count
26                                      (tDistInit ++ tDistFinal ++ tDistMid)
27                     eDist_count' = V.unsafeAccum (+) eDist_count eDistMid
28
29                 in (HMM m n tDist_count' eDist_count' stateMap observationMap)
30
31               ) (HMM m n tDist_initCount eDist_initCount stateMap observationMap)
32               corpusV
```

The initial counts for transition and emission probabilities are set up in line 17 and 19. These counts are used as starting values for folding over the corpus from line 21 to 32. Notice that multiple occurrences of the same sentences are considered as well, therefore the multiplication with $h(v)$ for an observation sequence v in for example line eight in the Baum-Welch algorithm drops out. In line 25 and line 27 the counts are updated. In form of a list of pairs *tDistInit*, *tDistMid*, *eDistMid* and *tDistFinal* contain information about at which position in the count-vectors which probability has to be added and are calculated for each sentence using the function *calcPartCounts*. Because Haskell is lazy, values inside the two vectors *tDist_count* and *eDist_count* are not updated immediately which leads to a lot of thunks. To avoid the fast increase of memory the strictness annotation *!* in line 21 is added.

```
33         calcPartCounts hmm osV = calcPartCounts' hmm osV (V.length osV)
34           where calcPartCounts' (HMM m n tDist eDist _ _) osV l =
35                   let forwardProb = L.foldl' (\prob state ->
36                                               let alphaProb = alpha 0 state
37                                                   betaProb  = beta  0 state
38                                               in  prob + alphaProb * betaProb)
39                                             (logFloat (0 :: Double)) [0..(m-1)]
40                       (tDistMid, eDistMid) =
41                         V.ifoldr' (\index word (tDistMid', eDistMid') ->
42                           if index == (l-1)
43                             then let eDistMid'New =
44                                               map (\(vt,qt) ->
45                                                 let iE = calcIndex qt n vt
46                                                     a  = alpha index qt
47                                                     b  = beta  index qt
```

```
48                                              v = (a * b) / forwardProb
49                                          in (iE,v)) [ (word,q)
50                                                     | q <- [0..(m-1)]
51                                                     ]
52                                  in   (tDistMid', eDistMid'New ++ eDistMid')
53                              else let tDistMid'New =
54                                         map (\(qt1,qt) ->
55                                           let iT = calcIndex qt (m+1) qt1
56                                               iE = calcIndex qt1 n
57                                                    (osV `V.unsafeIndex` (index+1))
58                                               a  = alpha index qt
59                                               b  = beta (index+1) qt1
60                                               t  = tDist `V.unsafeIndex` iT
61                                               e  = eDist `V.unsafeIndex` iE
62                                               v  = (a * b * t * e) / forwardProb
63                                           in (iT,v)) [ (q1,q)
64                                                      | q1 <- [0..(m-1)]
65                                                      , q  <- [0..(m-1)]
66                                                      ]
67                                       eDistMid'New =
68                                         map (\(vt,qt) ->
69                                           let iE = calcIndex qt n vt
70                                               a  = alpha index qt
71                                               b  = beta index qt
72                                               v  = (a * b) / forwardProb
73                                           in (iE,v)) [ (word,q)
74                                                      | q <- [0..(m-1)]
75                                                      ]
76                                  in  ( tDistMid'New ++ tDistMid',
77                                        eDistMid'New ++ eDistMid')) ([],[]) osV
78
79
80                          tDistInit = map (\(q1,sharp) ->
81                                             let iT = calcIndex sharp (m+1) q1
82                                                 a  = alpha 0 q1
83                                                 b  = beta 0 q1
84                                                 v  = (a * b) / forwardProb
85                                             in (iT,v)) [(q, m)
86                                                         | q <- [0..(m-1)]
87                                                         ]
88
89                          tDistFinal = map (\(sharp,qn) ->
90                                              let iT = calcIndex qn (m+1) sharp
91                                                  a  = alpha (l-1) qn
92                                                  b  = beta (l-1) qn
93                                                  v  = (a * b) / forwardProb
94                                              in (iT,v)) [(m, q)
95                                                          | q <- [0..(m-1)]
96                                                          ]
97                      in (tDistMid, eDistMid, tDistInit, tDistFinal)
98                      where alpha = -- local definition, memorized alpha
99                            beta  = -- local definition, memorized beta
```

The calculation of *tDistInit* and *tDistFinal* corresponds to line twelve to line 14 in the Baum-Welch algorithm. The two iterations over an observation sequence in line six to line eleven in the Baum-Welch algorithm are combined in one iteration in line 40 to line 77.
In line 35 the probability of the current observation sequence is calculated. This is not done by using the forward or backward algorithm directly, but by using the following observation:
$P(\mathbb{Q}_t = q_j, \mathbb{V} = v_1 ... v_n)$ from (30), page 19, was calculated by decomposing it for $1 \leq t < n$ as follows:

$$\begin{aligned} &P(\mathbb{Q}_t = q_j, \mathbb{V} = v_1 ... v_n) \\ &\quad = P(\mathbb{V}_1 = v_1, ..., \mathbb{V}_t = v_t, \mathbb{Q}_t = q_j) \cdot P(\mathbb{V}_{t+1} = v_{t+1}, ..., \mathbb{V} = v_n, \mathbb{L} = n \mid \mathbb{Q}_t = q_j) \\ &\quad = \alpha_t(q_j) \cdot \beta_t(q_j) \end{aligned}$$

By using Theorem 2.7 it can be seen that calculating $P(\mathbb{V} = v_1 ... v_n)$ can be achieved by summing over the equation above for all possible $q_j \in Q$ [JM09, p. 224] as it is done in line 35 for $t = 1$.

The advantage of calculating the probability of the observation sequence this way is that because the *alpha* and *beta* functions are memorized and defined locally in lines 98 and 99, values calculated for *alpha* and *beta* can be reused when calculating *tDistInit*, *tDistMid*, *eDistMid* and *tDistFinal*. The implementation of *mstep* which turns the two counts into two conditional probability distributions is not displayed here.

4.2 The HMM.Test module

As stated before, the *HMM.Test* module provides functions which help to ensure that HMM work correctly. Among others, the following functions are provided:

- *forwardBackwardCheck*
- *likelihoodCheck*

In Section 3 it was described that the forward and backward algorithms yield the same probability for same input. *forwardBackwardCheck hmm n* creates for the HMM *hmm* 100 random sentences of length n and checks if the *forward* and *backward* implementations yield the same results.
In the same way *likelihoodCheck hmm o p q* creates 100 random corpora which consist of p sentences of length o and checks for each corpus if the likelihood stays the same or increases after each of q iterations in the Baum-Welch algorithm.

An example usage could look as follows:

```
1  import HMM
2  import HMM.Test
3
4  main = do
5
6    case (uniHMM ([1..10] :: [Int]) ([1..6] :: [Int])) of
7      Nothing  -> putStrLn $ "HMM cannot be constructed from " ++
8                             "zero observations or zero states."
9      Just hmm -> do forwardBackwardCheck hmm 50
10                    likelihoodCheck hmm 4 10 5
```

4.3 The HMM module

As mentioned above, the *HMM* module exports all functions which are of interest to the end user: *uniHMM*, *randomHMM*, *forward*, *backward*, *baumWelch* and *likelihood*.

Referring to Example 3.2, performing 2000 iterations inside the Baum-Welch algorithm and calculating the probability of the observation sequence ($\mathbb{V}$ = Alice likes Alice) can be realized as follows:

```
1  import HMM
2
3  main = do
4
5    let states       = ["subject", "verb"]
6        observations = ["Alice", "likes", "sees", "him", "her"]
7        corpus       = [["Alice", "likes", "him"], ["Alice", "sees", "her"]]
8        iterations   = 2000
9
10   initHMM <- randomHMM states observations
11
12   case initHMM of
13     Nothing  -> putStrLn $ "HMM cannot be constructed from " ++
14                            "zero observations or zero states."
15     Just hmm -> case baumWelch hmm corpus iterations of
16                   Nothing         -> putStrLn "Corpus does not match HMM."
17                   Just trainedHMM -> print $ forward
18                                              trainedHMM ["Alice", "likes", "Alice"]
```

5 Empirical comparison of the quality of the bigram model and HMM

In the following the quality of the bigram model and the HMM is compared empirically on the basis of the corpus from Section 1:

Alice likes him Alice sees her

The quality is valuated by considering the likelihood of the models given the corpus and the probability sentences get assigned by the models. Sentences which to human understanding are grammatically right, these which are grammatically wrong, these which are and these which are not part of the corpus are regarded. Furthermore, for HMM different number of states and different initial probabilities are considered. The results can be reproduced by using the functions described in Section 4.

5.1 Quality of the bigram model

As mentioned in Section 1, conditional relative frequency yields the following conditional probability distribution:

$b(. \mid .)$	#	Alice	likes	sees	him	her
#	0	0	0	0	1	1
Alice	1	0	0	0	0	0
likes	0	0.5	0	0	0	0
sees	0	0.5	0	0	0	0
him	0	0	1	0	0	0
her	0	0	0	1	0	0

Table 5.1.1: Bigram model

The following table shows the probability for different sentences:

	Probability
Alice likes him	0.5
Alice sees her	0.5
Alice likes her	0
Alice sees him	0
Alice sees Alice	0
Alice likes Alice	0
Alice sees sees	0
sees Alice sees	0
sees sees sees	0

Table 5.1.2: Probabilities for the bigram model

As the high likelihood of $0.5 \cdot 0.5 = 0.25$ indicates, the bigram model matches the corpus perfectly: Both sentences in the corpus are assigned the probability of 0.5. However, all other sentences, including these which are grammatically correct such as *Alice likes her*, are valuated with zero.

5.2 Quality of the HMM with uniform initial probabilities

As with the bigram model the set of observations is the set of words which occur in the corpus. Assuming the one-element state set $Q = \{q_0\}$, an initial HMM with uniform distributed probabilities as described in Section 4.1.2 is displayed in Figure 5.2.1. Performing 2000 iterations of the Baum-Welch algorithm yields the trained HMM shown in Figure 5.2.2.

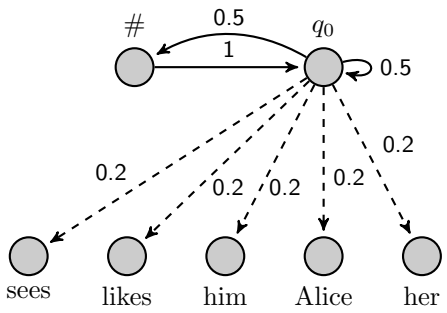


Figure 5.2.1: Initial HMM

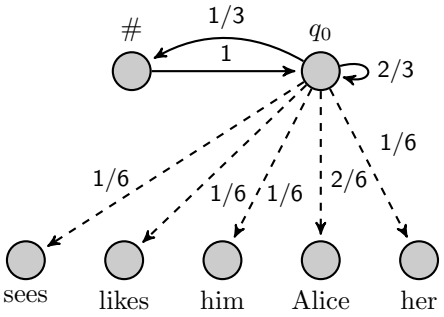


Figure 5.2.2: Trained HMM

The obtained transition and emission probabilities of the trained HMM stand to reason: For example the observation *Alice* occurs in the corpus consisting of six observation two times, therefore observing *Alice* in the only emitting state q_0 is done with a probability of $\frac{2}{6}$. In the same way the probability $t(\# \mid q_0) = \frac{1}{3}$ is reasonable: If all sentences in the corpus had the length 1, the probability of transitioning from q_0 to # would be 1. If the sentences had the length 2 the probability would be $\frac{1}{2}$ and with a length of 3 the probability would be $\frac{1}{3}$, because for sentences of length 3 two transitions from state q_0 to q_0 and one transition from state q_0 to # is made.

The following tables show the probabilities[9] assigned to the sentences by the two HMM.

	Probability
Alice likes him	$1 \cdot 10^{-3}$
Alice sees her	$1 \cdot 10^{-3}$
Alice likes her	$1 \cdot 10^{-3}$
Alice sees him	$1 \cdot 10^{-3}$
Alice sees Alice	$1 \cdot 10^{-3}$
Alice likes Alice	$1 \cdot 10^{-3}$
Alice sees sees	$1 \cdot 10^{-3}$
sees Alice sees	$1 \cdot 10^{-3}$
sees sees sees	$1 \cdot 10^{-3}$

Table 5.2.3: Probabilities for the initial HMM

	Probability
Alice likes him	$1.37 \cdot 10^{-3}$
Alice sees her	$1.37 \cdot 10^{-3}$
Alice likes her	$1.37 \cdot 10^{-3}$
Alice sees him	$1.37 \cdot 10^{-3}$
Alice sees Alice	$2.74 \cdot 10^{-3}$
Alice likes Alice	$2.74 \cdot 10^{-3}$
Alice sees sees	$1.37 \cdot 10^{-3}$
sees Alice sees	$1.37 \cdot 10^{-3}$
sees sees sees	$6.86 \cdot 10^{-4}$

Table 5.2.4: Probabilities for the trained HMM

Compared to Table 5.2.3 Table 5.2.4 shows that the higher probability of observing *Alice* results in a higher probability for sentence which contain *Alice* more often. However, the probabilities for sentences, whether they are grammatically correct or not, does not differ much. Furthermore, the likelihood of $1.88 \cdot 10^{-6}$ under the trained HMM is not much higher than the likelihood of $1 \cdot 10^{-6}$ under the initial HMM.
Training uniform HMM with more states leads to the same valuation of the given sentences and to the same likelihood.

5.3 Quality of the HMM with random initial probabilities

In the following, HMM with randomly distributed probabilities as described in Section 4.1.2 are considered.
Regardless of the exact initial probabilities, setting $|Q| = 1$ and performing 2000 iterations of the Baum-Welch algorithm always yields the HMM described in Figure 5.2.2.
In the case of $|Q| = 2$ the obtained HMM after training depends on the initial probabilities. Table 5.3.1 shows the number of times an HMM got obtained after training hundred random HMM and the corresponding likelihood. Because the names of the states are not important for calculating

[9] In the following, probabilities are rounded off to the second decimal place.

the probability of a sentence, checking if an HMM is already observed in the list of HMM is done by comparing if all parameters are the same except the naming of states.

	Number of occurrences	Likelihood
HMM 1	39	$2.44 \cdot 10^{-4}$
HMM 2	32	$2.44 \cdot 10^{-4}$
HMM 3	26	$2.44 \cdot 10^{-4}$
HMM 4	2	$1.53 \cdot 10^{-5}$
HMM 5	1	$7.53 \cdot 10^{-6}$

Table 5.3.1: HMM obtained after training 100 random HMM

HMM 4 and HMM 5 have a lower likelihood and occur less frequently then HMM 1, HMM 2 and HMM 3 and thus they are not to be discussed here.

HMM 1 and HMM 2 read as follows:

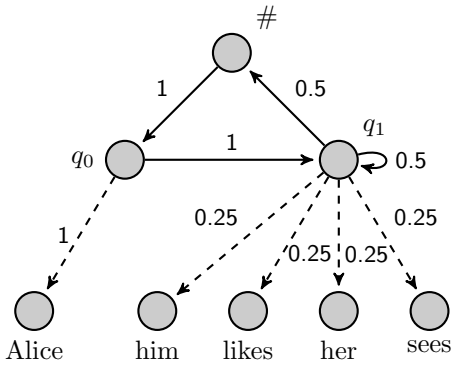


Figure 5.3.2: HMM 1

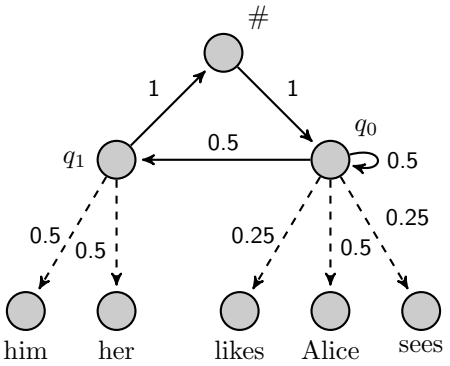


Figure 5.3.3: HMM 2

The idea of HMM 1 is to first produce the *Alice* observation and then to produce a sequence of *him*, *likes*, *sees* and *her* observations. In a similar way the HMM 2 at first produces a sequence of *likes*, *Alice* and *sees* and then adds either *him* or *her*.
As can be seen in the following two tables, this allows to assign high probabilities to grammatically correct sentences which are not part of the corpus.

	Probability
Alice likes him	$1.56 \cdot 10^{-2}$
Alice sees her	$1.56 \cdot 10^{-2}$
Alice likes her	$1.56 \cdot 10^{-2}$
Alice sees him	$1.56 \cdot 10^{-2}$
Alice sees Alice	0
Alice likes Alice	0
Alice sees sees	$1.56 \cdot 10^{-2}$
sees Alice sees	0
sees sees sees	0

Table 5.3.4: Probabilities for HMM 1

	Probability
Alice likes him	$1.56 \cdot 10^{-2}$
Alice sees her	$1.56 \cdot 10^{-2}$
Alice likes her	$1.56 \cdot 10^{-2}$
Alice sees him	$1.56 \cdot 10^{-2}$
Alice sees Alice	0
Alice likes Alice	0
Alice sees sees	0
sees Alice sees	0
sees sees sees	0

Table 5.3.5: Probabilities for HMM 2

HMM 3 is shown in figure 5.3.6.

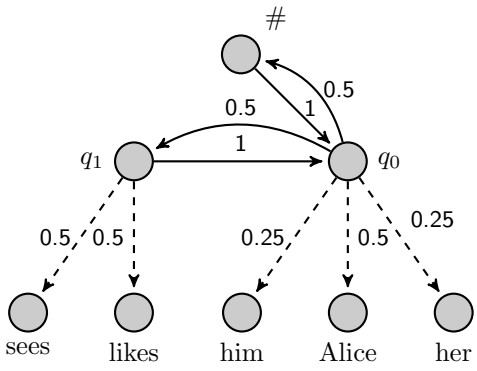


Figure 5.3.6: HMM 3

Substituting q_0 by *noun* and q_1 by *verb* one yields exactly the HMM intuitively constructed in Example 3.2. This allows more grammatically correct sentences to be valuated better compared to HMM 1 and HMM 2:

	Probability
Alice likes him	$1.56 \cdot 10^{-2}$
Alice sees her	$1.56 \cdot 10^{-2}$
Alice likes her	$1.56 \cdot 10^{-2}$
Alice sees him	$1.56 \cdot 10^{-2}$
Alice sees Alice	$3.13 \cdot 10^{-2}$
Alice likes Alice	$3.13 \cdot 10^{-2}$
Alice sees sees	0
sees Alice sees	0
sees sees sees	0

Table 5.3.7: Probabilities for HMM 3

Another interesting HMM occurs if the number of states in Q is set to five. Generating 100 random HMM and performing 200 iterations of the Baum-Welch algorithm, the following HMM is obtained 41 times:

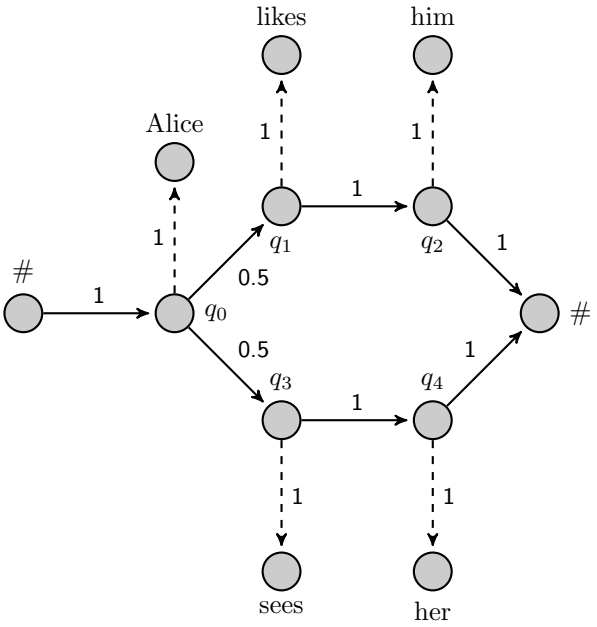


Figure 5.3.8: HMM equivalent to the bigram model shown in Table 5.1.1

Note that for reasons of space the start and final state # appears twice.
Each of the states in this HMM is responsible for producing exactly one observation and each of the observations in the corpus is emitted by exactly one state. Renaming the states by the observation they produce it is obvious that the conditional probability distribution of the bigram model in Table 5.1.1 is represented on the level of states in the HMM. In this sense this HMM is equivalent to the bigram model. Therefore it also has a likelihood of 0.25 and matches the corpus perfectly but also neglects the validity of other grammatically correct sentences.

Further increase of the number of states leads to a higher share of HMM obtained after the training under which the likelihood is 0.25.

5.4 Comparison

The examples above show that it is probably appropriate to prefer random initial probabilities over uniformly distributed initial probabilities. Furthermore, an HMM with a low complexity, for example only consisting of one state, leads to a low likelihood and a greater mistake concerning the level of abstraction (see Table 5.2.4). On the other hand a model which is too complex, for example see Figure 5.3.8, leads to high likelihood but is restricted too much to the corpus. The optimal complexity in the examples above amounts to $|Q| = 2$. HMM 3 (see Figure 5.3.6) shows that HMM are able to capture the linguistic rules behind a corpus better than the bigram model. To determine a suitable number of states and a specific HMM the method of cross validation [St3] can be used. At this different HMM are tested against another corpus of similar shape. For example testing HMM 1 (Figure 5.3.2), HMM 2 (Figure 5.3.3) and HMM 3 (Figure 5.3.6) against a corpus consisting of sentences like *Alice likes Alice*, the likelihood of HMM 3 under this corpus will be the highest because *Alice likes Alice* is assigned a higher probability by HMM 3 than by HMM 1 and HMM 2.

6 Further usage of HMM in natural language processing

Besides calculating the probability of a sentence, hidden Markov models can be used for further tasks in natural language processing.

One important task is *part-of-speech tagging* [MS99, p. 341]. At this each word in a sentence is assigned a corresponding part-of-speech respectively a tag, for example *noun* or *verb*. Regarding the set of states Q of the HMM $H = (Q, V, \#, t, e)$ as the set of syntactic categories or tags, HMM can be used to find the most likely tag sequence $q_1...q_n \in Q^+$ for a given sentence $v_1...v_n \in V^+$: [JM09, p. 173] [MS99, p. 332]

$$\begin{aligned} q_1...q_n &= \underset{q'_1...q'_n}{\arg\max}\, P(\mathbb{Q} = q'_1...q'_n \mid \mathbb{V} = v_1...v_n) \\ &= \underset{q'_1...q'_n}{\arg\max}\, \frac{P(\mathbb{Q} = q'_1...q'_n, \mathbb{V} = v_1...v_n)}{P(\mathbb{V} = v_1...v_n)} \\ &= \underset{q'_1...q'_n}{\arg\max}\, P(\mathbb{Q} = q'_1...q'_n, \mathbb{V} = v_1...v_n) \end{aligned}$$

Considering the HMM from Example 3.2, the most likely state sequence for *Alice likes Alice* is *noun verb noun*:

	Probability
Alice likes Alice, noun noun noun	0
Alice likes Alice, noun noun verb	0
Alice likes Alice, noun verb noun	$3.125 \cdot 10^{-2}$
Alice likes Alice, noun verb verb	0
Alice likes Alice, verb noun noun	0
Alice likes Alice, verb noun verb	0
Alice likes Alice, verb verb noun	0
Alice likes Alice, verb verb verb	0

Table 6.0.1: Probabilities for *Alice likes Alice* and all possible state sequences

Therefore the tags to the specific words in this sentence given this simple HMM are known. For more complex cases, the *Viterbi algorithm* can be used to efficiently calculate the state sequence $q_1...q_n$ which maximizes $P(\mathbb{Q} = q_1...q_n, \mathbb{V} = v_1...v_n)$ [MS99, p. 332].

A further area of application of HMM is in *speech recognition*, as described in [You06, pp. 3-5]: In a step of preparation the audio signal is disassembled into a sequence of feature vectors. For simplicity, the task of *isolated word recognition* is considered, which means that a given sequence of feature vectors is known to belong to exactly one word. The procedure then is to construct a HMM for each word one wants to recognize, whereas the set of observations of each HMM consists of all possible feature vectors. Knowing different sequences of feature vectors for one specific word, the HMM for this word can be trained trusting this HMM to assign a sequence of feature vectors which represents the current word a high probability afterwards. The task of recognizing a word from a given sequence of feature vectors then is done by finding the HMM which assigns the sequence the highest probability.

References

[BPSW70] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, page 164–171, 1970.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[JM09] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Prentice Hall, Upper Saddle River, N.J., second edition, 2009. ISBN 9780135041963.

[Les12] Roman Leshchinskiy. Data.Vector.Unboxed. Haskell module, Hackage, `http://hackage.haskell.org/packages/archive/vector/0.10.0.1/doc/html/Data-Vector-Unboxed.html`, 2012. [Online; accessed 12-June-2013].

[Man06] Tobias Mann. Numerically Stable Hidden Markov Model Implementation. University Lecture, University of Washington, Genome Sciences Department, `http://bozeman.genome.washington.edu/compbio/mbt599_2006/hmm_scaling_revised.pdf`, 2006. [Online; accessed 9-June-2013].

[Mil13] Ivan Miljenovic. Sourcegraph. Haskell package, Hackage, `http://hackage.haskell.org/package/SourceGraph`, 2013. [Online; accessed 12-June-2013].

[MS99] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999. ISBN 9780262133609.

[Pfe13] Avi Pfeffer. CS181 Lecture 17 — Hidden Markov Models. University Lecture, Harvard University, `http://www.seas.harvard.edu/courses/cs181/files/lecture17-notes.pdf`, 2013. [Online; accessed 4-June-2013].

[Rab89] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[St3] Torsten Stüber. Maschinelles Lernen in der Sprachverarbeitung. University Lecture, Technische Universität Dresden, Faculty of Computer Science, Chair of Foundations of Programming, 2013.

[Vog10] Heiko Vogler. Algorithmen, Datenstrukturen und Programmierung. University Lecture, Technische Universität Dresden, Faculty of Computer Science, Chair of Foundations of Programming, 2010.

[Vog12] Heiko Vogler. Maschinelles Übersetzen natürlicher Sprachen. University Lecture, Technische Universität Dresden, Faculty of Computer Science, Chair of Foundations of Programming, 2012.

[wnt10] wren ng thornton. Data.Number.LogFloat. Haskell module, Hackage, `http://hackage.haskell.org/packages/archive/logfloat/0.12.1/doc/html/Data-Number-LogFloat.html`, 2010. [Online; accessed 18-June-2013].

[You06] Steve Young. The HTK Book. Documentation for the Hidden Markov Model Toolkit, Cambridge University Engineering Department, `http://htk.eng.cam.ac.uk/docs/docs.shtml`, 2006. [Online; accessed 14-June-2013].