Markov chains and Hidden Markov Models

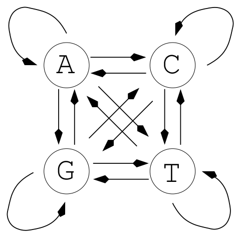
We will discuss:

- Hidden Markov Models (HMMs)
- Algorithms: Viterbi, forward, backward, posterior decoding
- Baum-Welch algorithm

Markov chains

Remember the concept of *Markov chains.* It is a probabilistic model in which the probability of one symbol depends on the probability of its predecessor.

Example.



- Circles = *states*, e.g. with names A, C, G and T.
- Arrows = possible *transitions* , each labeled with a *transition probability* a_{st}. Let x_i denote the state at time i. Then $a_{st} := P(x_{i+1} = t \mid x_i = s)$ is the conditional probability to go to state t in the next step, given that the current state is s.

Definition.
A (time-homogeneous) *Markov chain* (of order 1) is a system (Q, A) consisting of a finite set of *states* $Q = \{s_1, s_2, \dots, s_n\}$ and a *transition matrix* $A = \{a_{st}\}$ with $\sum_{t \in Q} a_{st} = 1$ for all $s \in Q$ that determines the probability of the transition $s \rightarrow t$ by

$$P(x_{i+1} = t \mid x_i = s) = a_{st}.$$

At any time i the Markov chain is in a specific state x_i, and at the tick of a clock the chain changes to state x_{i+1} according to the given transition probabilities.

Remarks on terminology.

- *Order 1* means that the transition probabilities of the Markov chain can only "remember" *1* state of its history. Beyond this, it is *memoryless*. The "memorylessness" condition is a very important. It is called the *Markov property*.

- The Markov chain is *time-homogenous* because the transition probability

$$P(x_{i+1} = t \mid x_i = s) = a_{st}.$$

does not depend on the time parameter i.

Example.
Weather in Tübingen, daily at midday: Possible states are “rain”, “sun”, or “clouds”.

Transition probabilities:

	R	S	C
R	.5	.1	.4
S	.2	.5	.3
C	.3	.3	.4

Note that all *rows* add up to 1.

Weather: `...rrrrrrccsssssscscscccrrcrcssss...`

Given a sequence of states $s_1, s_2, s_3, \dots, s_L$. What is the probability that a Markov chain $x = x_1, x_2, x_3, \dots, x_L$ will step through precisely this sequence of states? We have

$$\begin{aligned}
&P(x_L = s_L, x_{L-1} = s_{L-1}, \dots, x_1 = s_1) \\
&\quad = P(x_L = s_L \mid x_{L-1} = s_{L-1}, \dots, x_1 = s_1) \\
&\quad\quad \cdot P(x_{L-1} = s_{L-1} \mid x_{L-2} = s_{L-2}, \dots, x_1 = s_1) \\
&\quad\quad \vdots \\
&\quad\quad \cdot P(x_2 = s_2 \mid x_1 = s_1) \\
&\quad\quad \cdot P(x_1 = s_1)
\end{aligned}$$

using the “expansion”

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \iff P(A \cap B) = P(A \mid B) \cdot P(B).$$

Now, we make use of the fact that

$$P(x_i = s_i \mid x_{i-1} = s_{i-1}, \dots, x_1 = s_1) = P(x_i = s_i \mid x_{i-1} = s_{i-1})$$

by the Markov property. Thus

$$\begin{aligned}
&P(x_L = s_L, x_{L-1} = s_{L-1}, \dots, x_1 = s_1) \\
&\quad = P(x_L = s_L \mid x_{L-1} = s_L - 1) \\
&\quad\quad \cdot P(x_{L-1} = s_{L-1} \mid x_{L-2} = s_{L-2}) \\
&\quad\quad \cdot \ldots \cdot P(x_2 = s_2 \mid x_1 = s_1) \cdot P(x_1 = s_1) \\
&\quad = P(x_1 = s_1) \prod_{i=2}^{L} a_{s_{i-1} s_i}.
\end{aligned}$$

Hence:

The probability of a path is the product of the probability of the initial state and the transition probabilities of its edges.

Modeling the begin and end states

A Markov chain *starts* in state x_1 with an initial probability of $P(x_1 = s)$. For simplicity (i.e., uniformity of the model) we would like to model this probability as a transition, too.

Therefore we add a *begin state* to the model that is labeled 'b'. We also impose the constraint that $x_0 = \mathtt{b}$ holds. Then:

$$P(x_1 = s) = a_{\mathtt{b}s}\,.$$

This way, we can store all probabilities in one matrix and the "first" state x_1 is no longer special:

$$P(x_L = s_L, x_{L-1} = s_{L-1}, \ldots, x_1 = s_1) = \prod_{i=1}^{L} a_{s_{i-1}s_i}\,.$$

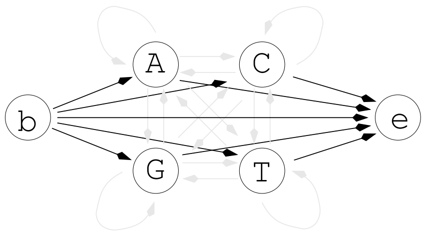
Similarly, we explicitly model the *end* of the sequence of states using an *end state* 'e'. Thus, the probability that the Markov chain stops is

$$P(x_L = t) = a_{x_L \mathtt{e}}.$$

if the current state is t.

We think of b and e as *silent* states, because they do not correspond to letters in the sequence. (More applications of silent states will follow.)

Example:



```
# Markov chain that generates CpG islands
# (Source: DEMK98, p 50)
# Number of states:
6
# State labels:
A,  C,  G,  T,  *=b,  +=e
# Transition matrix:
0.1795 0.2735 0.4255 0.1195 0 0.002
0.1705 0.3665 0.2735 0.1875 0 0.002
0.1605 0.3385 0.3745 0.1245 0 0.002
0.0785 0.3545 0.3835 0.1815 0 0.002
0.2495 0.2495 0.2495 0.2495 0 0.002
0.0000 0.0000 0.0000 0.0000 0 1.000
```

Determining the transition matrix

How do we find transition probabilities that explain a given set of sequences best?

The transition matrix A^+ for DNA that comes from a CpG-island, is determined as follows:

$$a^+_{st} = \frac{c^+_{st}}{\sum_{t'} c^+_{st'}},$$

where c_{st} is the number of positions in a training set of CpG-islands at which state s is followed by state t. We can calculate these counts in a single pass over the sequences and store them in a $\Sigma \times \Sigma$ matrix.

We obtain the matrix A^- for non-CpG-islands from empirical data in a similar way.

In general, the matrix of transition probabilities is not symmetric.

Two examples of Markov chains.

```
# Markov chain for CpG islands
# (Source: DEMK98, p 50)
# Number of states:
6
# State labels:
A C G T * +
# Transition matrix:
.1795 .2735 .4255 .1195 0 0.002
.1705 .3665 .2735 .1875 0 0.002
.1605 .3385 .3745 .1245 0 0.002
.0785 .3545 .3835 .1815 0 0.002
.2495 .2495 .2495 .2495 0 0.002
.0000 .0000 .0000 .0000 0 1.000
```

```
# Markov chain for non-CpG islands
# (Source: DEMK98, p 50)
# Number of states:
6
# State labels:
A C G T * +
# Transition matrix:
.2995 .2045 .2845 .2095 0 .002
 .3215 .2975 .0775 .3015 0 .002
.2475 .2455 .2975 .2075 0 .002
.1765 .2385 .2915 .2915 0 .002
.2495 .2495 .2495 .2495 0 .002
.0000 .0000 .0000 .0000 0 1.00
```

Note the different values for CpG: $a^+_{\mathrm{CG}} = 0.2735$ versus $a^-_{\mathrm{CG}} = 0.0775$.

Testing hypotheses

When we have two models, we can ask which one explains the observation better.

Given a (short) sequence $x = (x_1, x_2, \dots, x_L)$. Does it come from a CpG-island (model$^+$)?

We have

$$P(x \mid \text{model}^+) \;=\; \prod_{i=0}^{L} a^+_{x_i x_{i+1}},$$

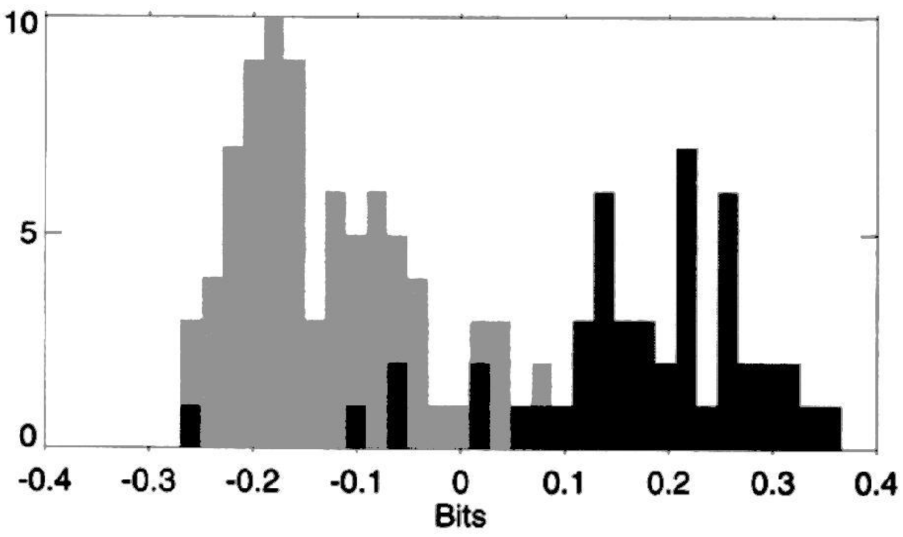
with $x_0 = \mathtt{b}$ and $x_{L+1} = \mathtt{e}$. Similar for (model$^-$).

To compare the models, we calculate the *log-odds ratio*:

$$S(x) \;=\; \log \frac{P(x \mid \text{model}^+)}{P(x \mid \text{model}^-)} \;=\; \sum_{i=0}^{L} \log \frac{a^+_{x_{i-1} x_i}}{a^-_{x_{i-1} x_i}}.$$

Then this ratio is normalized by the length of x. This resulting *length-normalized log-odds score* $S(x)/|x|$ can be used to classify x. The higher this score is, the higher the probability is that x comes from a CpG-island.

The histogram of the length-normalized scores for the sequences from the training sets for A^+ and A^- shows that $S(x)/|x|$ is indeed a good classifier for this data set. (Since the base two logarithm was used, the unit of measurement is called "bits".)



Example.
Weather in Tübingen, daily at midday: Possible states are rain, sun or clouds.

Types of questions that the Markov chain model can answer:

If it is sunny today, what is the probability that the sun will shine for the next seven days?

And what is more unlikely: 7 days sun or 8 days rain?

Hidden Markov Models (HMMs)

Motivation: Question 2, how to *find* `CpG`-islands in a long sequence?

We could approach this using Markov Chains and a "window technique": a window of width w is moved along the sequence and the score (as defined above) is plotted. However the results are somewhat unsatifactory: It is hard to determine the boundaries of CpG-islands, and which window size w should one choose? . . .

The basic idea is to relax the tight connection between "states" and "symbols". Instead, every state can "emit" every symbol. There is an emission probability $e_k(b)$ for each state k and symbol b.

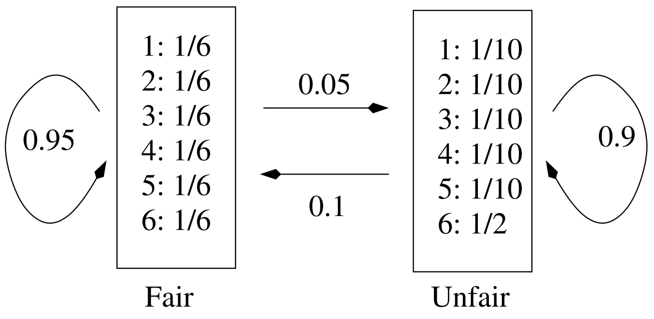
However, when we do so, we no longer know for sure the state from which an observed symbol was emitted. In this sense, the Markov model is hidden; hence the name.

Definition.
Am *HMM* is a system $M = (\Sigma, Q, A, e)$ consisting of

- an *alphabet* Σ,
- a set of *states* Q,
- a matrix $A = \{a_{kl}\}$ of *transition probabilities* a_{kl} for $k, l \in Q$, and
- an *emission probability* $e_k(b)$ for every $k \in Q$ and $b \in \Sigma$.

Example: fair/loaded die

An *occasionally dishonest casino* uses two dice, a *fair* and a *loaded* one:



A casino guest only observes the *numbers* rolled:
`6 4 3 2 3 4 6 5 1 2 3 4 5 6 6 6 3 2 1 2 6 3 4 2 1 6 6 ...`

However, which *die* was used remains hidden:
`F F F F F F F F F F F F U U U U U F F F F F F F F F F ...`

Generation of simulated data

HMMs, like Markov chains, are related to stochastic regular grammars and finite automata. Here is how we can generate a random sequence using an HMM:

Algorithm: Random sequence from HMM

- Start in state 0.
- While we have not re-entered state 0:
  - Choose a new *state* according to the transition probabilities.
  - Choose a *symbol* using the emission probabilities, and report it.

Example.
Here the fair/loaded HMM was used to generate a sequence of states and symbols:

```
States : FFFFFFFFFFFFFFUUUUUUUUUUUUUUUUUUFFFFFFFFFFUUUUUUUUUUUUUFFFF
Symbols: 24335642611341666666526562426612134635535566462666636664253

States : FFFFFFFFFFFFFFFFFFFFFFFFFFFUUUUUUUFFUUUUUUUUUUUUUUFFFFFFFFF
Symbols: 35246363252521655615445653663666511145445656621261532516435

States : FFUUUUUUUU
Symbols: 5146526666
```

Questions.

- Given an HMM and a sequence of states and symbols. What is the *probability* to get this sequence?
- Given an HMM and a sequence of symbols. Can we *reconstruct* the corresponding sequence of states, assuming that the sequence was generated using the HMM?

Probability for given states and symbols

Definitions.

- A *path* $\pi = (\pi_1, \pi_2, \dots, \pi_L)$ in an HMM $M = (\Sigma, Q, A, e)$ is a sequence of states $\pi_i \in Q$.
- Given a sequence of *symbols* $x = (x_1, \dots, x_L)$ and a *path* $\pi = (\pi_1, \dots, \pi_L)$ through M. Then the *joint probability* is:

$$P(x, \pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}},$$

with $\pi_{L+1} = 0$.

Schematically,

$$\text{begin} \rightarrow \underset{a_{0,\pi_1}}{\text{transition}} \rightarrow \underset{e_{\pi_1}(x_1)}{\text{emission}} \rightarrow \underset{a_{\pi_1,\pi_2}}{\text{transition}} \rightarrow \underset{e_{\pi_2}(x_2)}{\text{emission}} \rightarrow \cdots \rightarrow \underset{a_{\pi_L,0}}{\text{transition}} \rightarrow \text{end}$$

All we need to do is multiply these probabilities .

This answers the first question. However, usually we do *not know* the path π through the model! That information is *hidden*.

The decoding problem

The *decoding problem* is the following: We have observed a sequence x of symbols that was generated by an HMM and we would like to “decode” the sequence of states from it.

The most probable path. To solve the decoding problem, we want to determine the path π^* that maximizes the probability of having generated the sequence x of symbols, that is:

$$\pi^* = \arg\max_{\pi} \Pr(\pi \mid x) = \arg\max_{\pi} \Pr(\pi, x).$$

For a sequence of n symbols there are $|Q|^n$ possible paths, therefore we cannot solve the problem by full enumeration.

Luckily, the “most probable path” π^* can be computed by *dynamic programming*. The recursion involves the following entities:

Definition. Given a prefix $(x_1, x_2, \dots, x_i)$ of the sequence x which is to be decoded. Then let $(\pi_1^*, \pi_2^*, \dots, \pi_i^*)$ be a *path of states* with $\pi_i^* = s$ which maximizes the probability that the HMM followed theses states and emitted the symbols $(x_1, x_2, \dots, x_i)$ along its way. That is,

$$(\pi_1^*, \dots, \pi_i^*) = \arg\max \left\{ \prod_{k=1}^{i} a_{\pi_{i-1},\pi_i} e_{\pi_i}(x_i) \;\middle|\; (\pi_1, \dots, \pi_i) \in Q^i, \pi_i = s, \pi_0 = 0 \right\}.$$

Also, let $V(s, i)$ denote the *value* of this maximal probability. These are sometimes called *Viterbi variables*.

Clearly we can store the values $V(s,i)$ in a $Q \times [1\ ..\ L]$ dynamic programming matrix.

Initialization.
Every path starts at state 0 with probability 1. Hence, the *initialization* for $i = 0$ is $V(0,0) = 1$, and $V(s,0) = 0$ for $s \in Q \setminus \{0\}$.

Recursion.
Now for the *recurrence formula*, which applies for $i = 1, \dots, L$. Assume that we know the most likely path for $x_1, \dots, x_{i-1}$ under the additional constraint that the last state is s, for all $s \in Q$. Then we obtain the most likely path to the i-th state t by maximizing the probability $V(s, i-1)a_{st}$ over all predecessors $s \in Q$ of t. To obtain $V(t,i)$ we also have to multiply by $e_t(x_i)$ since we have to emit the given symbol x_i.

That is, we have

$$V(t,i) = \max\{\ V(s,i-1)a_{st} \mid s \in Q\ \} \cdot e_t(x_i)$$

for all $t \in Q$. (Again, note the use of the Markov property!)

Termination.
In the last step, we enter state 0 but do not emit a symbol. Hence $P(x, \pi^*) = \max\{\ V(s,L)a_{s,0} \mid s \in Q\ \}$.

Viterbi algorithm

Input:	HMM $M = (\Sigma, Q, A, e)$
	and symbol sequence x
Output:	Most probable path π^*.
Initialization ($i = 0$)*:*	$V(0,0) = 1$, $V(s,0) = 0$ for $s \in Q \setminus \{0\}$.
Recurrence:	
For $i = 1, \dots, L$, $t \in Q$:	$V(t,i) = e_t(x_i) \max\{\ V(s,i-1)a_{s,t} \mid s \in Q\ \}$
	$T(t,i) = \arg\max\{\ V(s,i-1)a_{s,t} \mid s \in Q\ \}$
Termination ($i = L+1$*):*	$P(x,\pi^*) = \max\{\ V(s,L)a_{s,0} \mid s \in Q\ \}$
	$\pi^*_L = \arg\max\{\ V(s,L)a_{s,0} \mid s \in Q\ \}$
Traceback:	
For $i = L+1, \dots, 1$:	$\pi^*_{i-1} = T(\pi^*_i, i)$

The *running time* is $|Q|^2|L|$, as each entry of the $Q \times L$ matrix requires $|Q|$ calculations.

The fair/loaded HMM was used to *generate* a sequence of symbols and then the Viterbi-algorithm to *decode* the sequence. The result is:

```
True state:  FFFFFFFFFFFFFFUUUUUUUUUUUUUUUUUUFFFFFFFFFFUUUUUUUUUUUUUFFFF
Symbols:     24335642611341666666526562426612134635535566462666636664253
Viterbi:     FFFFFFFFFFFFFFUUUUUUUUUUUUUUUUFFFFFFFFFFFFUUUUUUUUUUUUUFFFF

True state:  FFFFFFFFFFFFFFFFFFFFFFFFFFFUUUUUUUFFUUUUUUUUUUUUUUFFFFFFFFF
Symbols:     35246363252521655615445653663666511145445656621261532516435
Viterbi:     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

True state:  FFUUUUUUUU
Symbols:     5146526666
Viterbi:     FFFFFFUUUU
```

Algorithmic tasks for HMMs

Let M be an HMM and x be a sequence of symbols.

1. Determine the sequence of states through M which has the highest probability to generate x: *Viterbi algorithm*
2. Determine the probability $P(x)$ that M generates x: *forward algorithm* or *backward algorithm*
3. Assuming that x was generated by M, determine the conditional probability that M was in state s when it generated the i-th symbol x_i: *Posterior decoding*, done by a combination of the *forward and backward algorithm*
4. Given x and perhaps some additional sequences of symbols, adjust the transition and emission probabilities of M such that it explains best these observations: E.g., *Baum-Welch algorithm*

Forward algorithm

We have already seen a closed formula for the probability $P(x,\pi)$ that M generated x using the path π. Summing over all possible paths, we obtain the probability that M generated x:

$$P(x) = \sum_{\pi} P(x,\pi)\,.$$

Calculating this sum is done by “replacing the max with a sum” in the Viterbi algorithm.

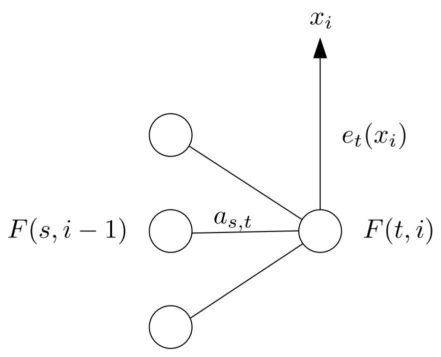
Formally, we define the *forward-variable*:

$$F(s,i) = \Pr(x_1 \dots x_i, \pi_i = s),$$

This is the probability that the sequence generated by the HMM has the sequence $(x_1,\dots,x_i)$ as a prefix, and the

i-th state is $\pi_i = s$. Using the distributive law (and a similar reasoning as for $\Pr(x,\pi)$), we obtain the recursion

$$F(t,i) = e_t(x_i) \cdot \sum_{s \in Q} F(s,i-1) a_{s,t}\,.$$



The resulting algorithm is actually simpler, since we do not have to maintain backtrace pointers:

Input:	HMM $M = (\Sigma, Q, A, e)$
	and symbol sequence x
Output:	Probability $\Pr(x)$.
Initialization ($i = 0$)*:*	$F(0,0) = 1$, $F(s,0) = 0$ for $s \in Q \setminus \{0\}$.
Recurrence:	
For all $i = 1 \dots L$, $t \in Q$:	$F(t,i) = e_t(x_i) \sum_{s \in Q} F(s,i-1) a_{st}$
Termination ($i = L+1$):	$\Pr(x) = \sum_{s \in Q} F(s,L) a_{s,0}$

Backward algorithm

Recall the definition of the *forward-variables*:

$$F(s,i) = \Pr(x_1 \dots x_i, \pi_i = s)\,.$$

This is the probability that the sequence generated by the HMM has the sequence $(x_1, \dots, x_i)$ as a *prefix*, and the i-th state is $\pi_i = s$.

For the *posterior decoding* (described later) we need to compute the conditional probability that M emitted x_i from state π_i, when it happened to generate x:

$$\Pr(\pi_i = s \mid x) = \frac{\Pr(\pi_i = s, x)}{\Pr(x)}\,.$$

Since $\Pr(x)$ is known by the forward algorithm, it suffices to calculate $\Pr(\pi_i = s, x)$. We have

$$\begin{aligned} \Pr(x, \pi_i = s) &= \Pr(x_1, \dots, x_i, \pi_i = s) \Pr(x_{i+1}, \dots, x_L \mid x_1, \dots, x_i, \pi_i = s) \\ &= \Pr(x_1, \dots, x_i, \pi_i = s) \Pr(x_{i+1}, \dots, x_L \mid \pi_i = s)\,, \end{aligned}$$

using the Markov property in the second equation.

Hence, if we define the *backward-variables*:

$$B(s,i) = \Pr(x_{i+1} \dots x_L \mid \pi_i = s)\,,$$

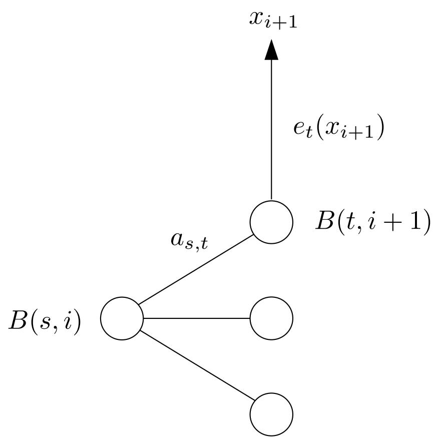
then

$$\begin{aligned}\Pr(x, \pi_i = s) &= \Pr(x_1, \ldots, x_i, \pi_i = s) \Pr(x_{i+1}, \ldots, x_L \mid \pi_i = s) \\ &= F(s, i) B(s, i) .\end{aligned}$$

$B(s, i)$ is the probability that the HMM will generate the sequence $(x_{i+1}, \ldots, x_L)$ “when it is started in state π_i”, until it reaches state 0. It is computed by the *backward algorithm*.

Input:	HMM $M = (\Sigma, Q, A, e)$
	and symbol sequence x
Output:	Probability $\Pr(x)$.
Initialization ($i = L$)*:*	$B(s, L) = a_{s,0}$ for all $s \in Q$.
Recurrence:	
For all $i = L-1 \ldots 1$, $s \in Q$:	$B(s, i) = \sum_{t \in Q} a_{st} e_t(x_{i+1}) B(t, i+1)$
Termination ($i = 0$*):*	$\Pr(x) = \sum_{t \in Q} a_{0,t} e_t(x_1) B(t, 1)$

The reasoning behind the recurrences is similar to the forward algorithm.



Usually one is mainly interested in the backward *variables*, since $\Pr(x)$ can be computed by the forward algorithm as well.

Posterior decoding

The probabilities

$$\Pr(\pi_i = s \mid x), \quad i = 1, \ldots, L, \, s \in Q$$

are called *posterior probabilities*, because they are determined *after* observing the (random) sequence x. They are conditional probabilities where we have a partial knowledge of the actual outcome of a random experiment.

The sequence of states $\hat{\pi}$ obtained by posterior decoding is defined thus:

$$\hat{\pi}_i = \arg\max_{s \in Q} P(\pi_i = s \mid x) .$$

In other words, at every position we choose the most probable state for that position.

Pros and cons of posterior decoding:

- Posterior probabilities provide an alternative means to decode the observed sequence. It is often superior to the Viterbi algorithm, especially if there are *many paths* which have *almost the same probability* as the most probable one.
- Posterior decoding is useful if we are interested in *the state at a specific position i* and not in the whole sequence of states.
- Also the posterior *probabilities* can serve as a *quantitative measurement* of the *confidence* in the predicted states.
- Although the posterior decoding is defined by maximizing the probabilities at each state, *the path as a whole might not be taken very likely* by the HMM. If the transition matrix forbids some transitions (i.e., $a_{st} = 0$), then this decoding may even produce a sequence that is *not a valid path*, because its probability is 0 !

The design of Hidden Markov Models

How does one come up with an HMM?

First step: Determine its "*topology*", i.e. the number of states and how they are connected via transitions of non-zero probability. – This is kind of an art. We will discuss later in the lecture the design of different gene finding HMMs.

Second step: Set the *parameters*, i.e. the transition probabilities a_{st} and the emission probabilities $e_s(x)$.

We consider the second step. Given a set of example sequences. Our goal is to "train" the parameters of the HMM using the example sequences, that is, to set the parameters in such a way that the probability, with which the HMM generates the given example sequences, is maximized.

Training when the states are known

Let $M = (\Sigma, Q, A, e)$ be an HMM.

Given a list of sequences of symbols $x^{(1)}, x^{(2)}, \dots, x^{(n)}$ and a list of corresponding paths $\pi^{(1)}, \pi^{(2)}, \dots, \pi^{(n)}$ (e.g., DNA sequences with annotated `CpG`-islands, or gene models). We consider the probability that the HMM generates them "one after another".

We want to choose the parameters (A, e) of the HMM M *optimally*, such that:

$$(A, e) \;=\; \arg\max \left\{ \Pr_{M'}\left((x^{(1)}, \pi^{(1)}), \dots, (x^{(n)}, \pi^{(n)})\right) \;\middle|\; M' = (\Sigma, Q, A', e') \right\}$$

In other words, we want to determine the *maximum likelihood estimator* (*ML-estimator*) for the parameters (A, e).

Not surprisingly, it turns out (analytically) that the likelihood is maximized by the estimators

$$\tilde{a}_{st} := \frac{\bar{a}_{st}}{\sum_{t'} \bar{a}_{s,t'}} \qquad \text{and} \qquad \tilde{e}_s(b) := \frac{\bar{e}_s(b)}{\sum_{b'} \bar{e}_s(b')},$$

where

$$\bar{a}_{st} := \text{Observed number of transitions from state } s \text{ to state } t,$$
$$\bar{e}_s(b) := \text{Observed number of emissions of symbol } b \text{ in state } s.$$

(However this presumes that we have sufficient training data.)

Training the *fair/loaded* HMM

Given example data x and π:

```
Symbols x: 1 2 5 3 4 6 1 2 6 6 3 2 1 5
States pi: F F F F F F F U U U U F F F
```

State transitions:

$\bar{a}_{kl}$	0	F	U
0			
F			
U			

$\rightarrow$

$\tilde{a}_{kl}$	0	F	U
0			
F			
U			

Emissions:

$\bar{e}_k(b)$	1	2	3	4	5	6
0						
F						
U						

$\rightarrow$

$\tilde{e}_k(b)$	1	2	3	4	5	6
0						
F						
U						

Given example data x and π:

```
Symbols x: 1 2 5 3 4 6 1 2 6 6 3 2 1 5
States pi: F F F F F F F U U U U F F F
```

State transitions:

$\bar{a}_{kl}$	0	F	U
0	0	1	0
F	1	8	1
U	0	1	3

$\rightarrow$

$\tilde{a}_{kl}$	0	F	U
0	0	1	0
F	$\frac{1}{10}$	$\frac{8}{10}$	$\frac{1}{10}$
U	0	$\frac{1}{4}$	$\frac{3}{4}$

Emissions:

$\bar{e}_k(b)$	1	2	3	4	5	6		$\tilde{e}_k(b)$	1	2	3	4	5	6
F	3	2	1	1	2	1	$\rightarrow$	F	.3	.2	.1	.1	.2	.1
U	0	1	1	0	0	2		U	0	$\frac{1}{4}$	$\frac{1}{4}$	.0	.0	$\frac{1}{2}$

Pseudocounts

One common problem in training is *overfitting*. For example, if some possible transition (s,t) is never seen in the example data, then we will set $\tilde{a}_{st} = 0$ and the transition is then forbidden in the resulting HMM. Moreover, if a given state s is never seen in the example data, then $\tilde{a}_{st}$ resp. $\tilde{e}_s(b)$ is undefined for all t, b.

To solve this problem, we introduce *pseudocounts* r_{st} and $r_s(b)$ and *add* them to the observed transition resp. emission frequencies.

Small pseudocounts reflect “little pre-knowledge”, large ones reflect “more pre-knowledge”. The effect of pseudocounts can also be thought of as “smoothing” the model parameters using a background model (i. e., a *prior distribution*).

Training when the states are unknown

Now we assume we are given a list of sequences of symbols $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ and we do *not* know the list of corresponding paths $\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(n)}$, as is usually the case in practice.

Then the problem to choose the parameters (A,e) of the HMM M *optimally*, such that:

$$(A,e) \;=\; \arg\max \left\{ \Pr_{M'}\left(x^{(1)}, \ldots, x^{(n)}\right) \;\middle|\; M' = (\Sigma, Q, A', e') \right\}$$

is NP-hard and hence we cannot be solve it exactly in polynomial time.

Instead we approximate it with an EM (expectation-maximization) procedure known as the Baum-Welch algorithm. We start by explaining the general principle of the EM algorithm.

The expectation-maximization algorithm

The *expectation-maximization (EM)* algorithm is a general paradigm which covers many iterative algorithms for parameter estimation.

Assume that a statistical model is determined by *model parameters* θ. The model shall be adapted to explain the *observed data* with maximum likelihood. Depending on the nature of the problem, this can be a difficult task.

Sometimes it is easier to consider a related parameter estimation problem, in which the observed data are augmented by *missing data*. These are also called *missing information* or *latent data*. Changing the model this way is sometimes called *data augmentation*.

The EM algorithm formalizes an intuitive idea for obtaining parameter estimates when some of the data are missing. It repeats the following two steps until convergence:

'E' step: Estimate the missing information using the current model parameters.

'M' step: Optimize the model parameters using the estimated missing information.

This idea has been in use for many years before Orchard and Woodbury (1972) in their *missing information principle* provided the theoretical foundation of the underlying idea. The term EM was introduced by Dempster, Laird, and Rubin (1977) where proof of general results about the behavior of the algorithm was first given as well as a large number of applications.

Let us introduce a bit of notation. Denote the *observed* part of the data by x, and the *missing* information by y. (In our case, x represents the input sequences and y the paths generating the sequences. (A, e) are the parameters to be adapted.)

The aim is to find the model parameters θ maximizing the likelihood given the observed data or, equivalently, the log likelihood:

$$\arg\max_{\theta} \log P(x \mid \theta) = \log \sum_y P(x, y \mid \theta).$$

The EM algorithm shall be used to solve this optimization problem. The log likelihood for the observed data might be hard to deal with directly, thus we start by considering the conditional log likelihood for the observed data given the missing information:

$$\log P(x \mid y, \theta).$$

Note that formally the missing information acts just like additional model parameters which are excluded from optimization. The question is, what should we plug in for the missing information y?

As the name says, we do not have the missing information. But of course, using the EM scheme can make sense only if the model allows us to predict the missing information in some way. If we consider $\log P(x \mid y, \theta)$ as a function of y, we obtain the posterior probability for y, given the observed data x and some model parameters θ, using Bayes' theorem as follows:

$$P(y \mid x, \theta) = \frac{P(x, y \mid \theta)}{P(x \mid \theta)} = \frac{P(x \mid y, \theta) P(y \mid \theta)}{\sum_{y'} P(x \mid y', \theta) P(y' \mid \theta)}.$$

Note that this formulation only requires knowledge of the observation likelihood given the missing information, $P(x \mid y, \theta)$, and the probability of the missing information, $P(y \mid \theta)$.

The EM algorithm works by improving an initial estimate θ^0 of the model parameters until a convergence criterion is met.

Usually we stop if the change in the log-likelihood sinks under a certain threshhold. In this way, we can rely on an estimate θ^t from the previous round of the main loop, and use $P(y \mid \theta^t)$ as an estimate for the distribution of the missing information according to some maximum likelihood model parameters.

Since we do not know the missing information y exactly, it is not realistic to rely on $\log P(x \mid y, \theta)$ for only one particular assignment of the missing information.

Instead we consider the *expected value of the log likelihood* $\log P(x \mid y, \theta)$ of the observations x and the model parameters θ *according to the posterior distribution of the missing information y (which is estimated according to θ^t).*

Formally this means that we replace the log likelihood

$$\log P(x \mid \theta) = \log \sum_y P(y \mid x, \theta^t) P(x \mid y, \theta)$$

by

$$\mathrm{el}(x,\theta) := \sum_y P(y \mid x,\theta^t) \log P(x \mid y,\theta)$$

which is a different function (!) but more amenable to optimization.

A greedy (short-sighted) approach would be to choose θ' such that $\mathrm{el}(x \mid \theta')$ is greater than $\mathrm{el}(x \mid \theta^t)$. Let us try this out.

Remember that Bayes theorem yields:

$$P(y \mid x,\theta) = \frac{P(x,y \mid \theta)}{P(x \mid \theta)}$$

Hence we can write:

$$P(x,y \mid \theta) = P(y \mid x,\theta)P(x \mid y,\theta)$$

and hence,

$$\log P(x,y \mid \theta) = \log P(y \mid x,\theta) + \log P(x \mid y,\theta)$$

or

$$\log P(x \mid y,\theta) = \log P(x,y \mid \theta) - \log P(y \mid x,\theta)$$

Then

$$\sum_y P(y \mid x,\theta^t) \log P(x \mid y,\theta) = \; = \underbrace{\sum_y P(y \mid x,\theta^t) \log P(x,y \mid \theta)}_{(I)} - \underbrace{\sum_y P(y \mid x,\theta^t) \log P(y \mid x,\theta)}_{(II)}$$

Ignore the second sum on the right hand side (II) for a moment. We denote the first sum on the right hand side (I) with

$$Q(\theta,\theta^t) := \sum_y P(y \mid x,\theta^t) \log P(x,y \mid \theta).$$

Note that $Q(\theta,\theta^t)$ is the average of $\log P(x,y \mid \theta)$ over the posterior distribution of y obtained with the current parameters θ^t.

Then the (greedy) update formula for the model parameters takes the form:

$$\theta^{t+1} = \arg\max_\theta Q(\theta,\theta^t).$$

This update formula subsumes both steps of the EM algorithm: The updated model parameters θ^{t+1} are chosen such as to *maximize (M)* the *conditional expectation (E)* of the complete data log likelihood, where the observed data are given, and the missing information is estimated according to the previous parameter values θ^t.

The factors $P(y \mid x,\theta^t)$ are easily computed in the E-step. They are excluded from the optimization in the M-step. Thus we maximize a weighted sum of log likelihood functions, one for each outcome of the hidden variables y.

Next we look at the other summand (II) in the objective function.

Look at the difference between the new term and the old:

$$\sum_y P(y \mid x,\theta^t) \log P(y \mid x,\theta^t) - \sum_y P(y \mid x,\theta^t) \log P(y \mid x,\theta) = \sum_y P(y \mid x,\theta^t) \log \frac{P(y \mid x,\theta^t)}{P(y \mid x,\theta)}$$

Observe that this is just the relative entropy of $P(y \mid x, \theta^t)$ with respect to $P(y \mid x, \theta)$. It is well-known that the relative entropy is always non-negative (exercise) and zero if and only if $\theta = \theta^t$.

Hence it holds that

$$0 \le Q(\theta^{t+1} \mid \theta^t) - Q(\theta^t \mid \theta^t) \le \mathrm{el}(x, \theta^{t+1}) - \mathrm{el}(x, \theta^t)$$

with equality only if $\theta^{t+1} = \theta^t$. It follows that we will always make the difference positive and thus increase the likelihood of x under the new model, unless we have $\theta^{t+1} = \theta^t$.

One can show that under reasonable assumptions the EM algorithm will approach a local optimum of the objective function.

We now see how the EM algorithm looks like in the context of HMMs. It is known under the name Baum-Welch algorithm.

In the case of HMMs the missing information are the paths π.

$$Q(\theta, \theta^t) := \sum_{\pi} P(\pi \mid x, \theta^t) \log P(x, \pi \mid \theta).$$

For a given path each parameter in the model will appear a number of times in $P(x, \pi \mid \theta)$. We now need only to estimate those numbers averaging over all paths.

The Baum-Welch algorithm finds a locally optimal solution to the HMM training problem. It starts from an arbitrary initial estimate for (A, e) and the observed frequencies $\bar{A}$, $\bar{e}$. Then it applies a reestimation step until convergence or timeout.

- In each reestimation step, the forward and backward algorithm is applied (with the current model parameters) to each example sequence.
- Using the posterior state probabilities, one can calculate the expected emission frequencies. (In a loop we add for each training sequence a term to the current $\bar{e}$).
- Using the forward and backward variables, one can also calculate the expected transition frequencies. (In a loop we add for each training sequence a term to $\bar{A}$).
- The new model parameters are calculated as maximum likelihood estimates based on $\bar{A}$, $\bar{e}$.

Baum-Welch algorithm

Input:	HMM $M = (\Sigma, Q, A, e)$,
	training data $x^{(1)}, \dots, x^{(n)}$,
	pseudocounts r_{st} and $r_s(b)$ (if desired).
Output:	HMM $M' = (\Sigma, Q, A', e')$ with an improved score.
Initialization:	Pick some arbitrary model parameters (A, e).
	Set some initial "observation frequencies" $(\bar{A}, \bar{e})$.
Reestimation:	For each sequence $x^{(j)}$, $j = 1, \dots, n$:
	Calculate $F^{(j)}$ using the forward algorithm.
	Calculate $B^{(j)}$ using the backward algorithm.
	Update $\bar{A}$ and $\bar{e}$ using the posterior probabilities.
	Calculate new model parameters (A, e) from $(\bar{A}, \bar{e})$.
	Calculate the new log-likelihood of the model.
	Repeat.
Termination:	Stop if the change of log-likelihood becomes too small,
	or the maximal number of iterations is exceeded.

Updating $\bar{e}$: The probability that the model with parameters (A, e) was in state $\pi_i = s$ when it generated the symbol $x_i^{(j)}$ of $x^{(j)}$ is just the posterior state probability

$$\Pr(\pi_i^{(j)} = s \mid x^{(j)}) = \frac{F^{(j)}(s,i)B^{(j)}(s,i)}{\Pr(x^{(j)})}.$$

Thus we can derive the expected number of times b is output in state s, i.e. $\bar{e}_s(b)$ by

$$\sum_{j=1}^{n} \sum_{i:x_i^{(j)}=b} \frac{F^{(j)}(s,i)B^{(j)}(s,i)}{\Pr(x^{(j)})}.$$

Hence we can compute the new model parameter for the emission probabilities as: $\tilde{e}_s(b) := \frac{\bar{e}_s(b)}{\sum_{b'} \bar{e}_s(b')}$.

Updating $\bar{A}$: The probability that the model with parameters (A, e) stepped from state $\pi_i = s$ to state $\pi_{i+1} = t$ when it generated the symbols $x_i^{(j)}, x_{i+1}^{(j)}$ of $x^{(j)}$ is

$$\begin{aligned}\Pr(\pi_i = s, \pi_{i+1} = t \mid x) &= \Pr(x_1, \dots, x_i, \pi_i = s) \cdot \Pr(x_{i+1}, \pi_i = s, \pi_{i+1} = t) \cdot \Pr(x_{i+2}, \dots, x_L \mid \pi_{i+1} = t) \\ &= \frac{F(s,i)a_{st}e_t(x_{i+1})B(t,i+1)}{\Pr(x^{(j)})}.\end{aligned}$$

Thus we can derive the expected number $\bar{a}_{st}$ of times that a_{st} is used in the training data as follows:

$$\sum_{j=1}^{n} \sum_{i=0}^{|x^{(j)}|} \frac{F(s,i)a_{st}e_t(x_{i+1})B(t,i+1)}{\Pr(x^{(j)})}.$$

(Here we let $e_t(x_{|x^{(j)}|+1}) := 1$.) Again we can use those values to update the matrix A.

Remark: One can prove that the log-likelihood-score converges to a local maximum using the Baum-Welch-algorithm.

However, this doesn't imply that the parameters converge!

Local maxima can be avoided by considering many different starting points.

Additionally, any standard optimization approaches can also be applied to solve the optimization problem.