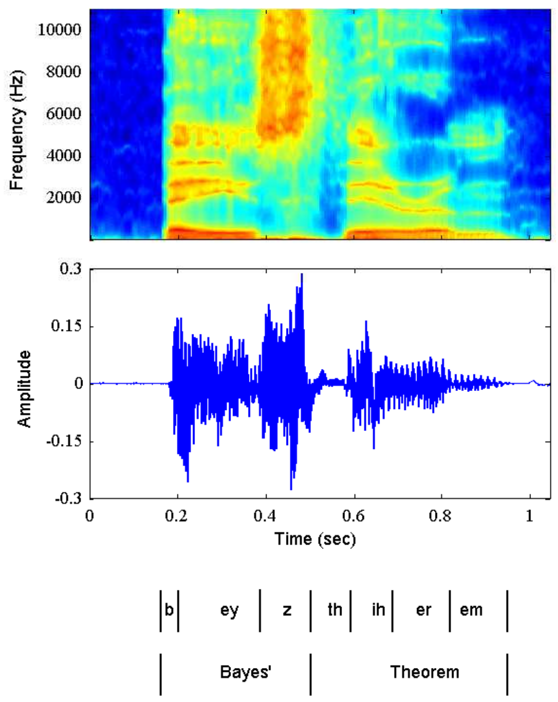

Figure 1: Examples of time-series data: speech and language.

16 Hidden Markov Models

Until now, we have exclusively dealt with data sets in which each individual measurement is independent and identically distributed (IID). That is, for two points $\mathbf{y}_1$ and $\mathbf{y}_2$ in our data set, we have $p(\mathbf{y}_1) = p(\mathbf{y}_2)$ and $p(\mathbf{y}_1, \mathbf{y}_2) = p(\mathbf{y}_1)p(\mathbf{y}_2)$ (for a fixed model). Time-series data consist of sequences of data that are not IID: they arise from a process that varies over time, and modeling the dynamics of this process is important.

16.1 Markov Models

Markov models are time series that have the Markov property:

$$P(s_t | s_{t-1}, s_{n-2}, ..., s_1) = P(s_t | s_{t-1}) \tag{1}$$

where s_t is the state of the system at time t. Intuitively, this property says the probability of a state at time t is competely determined by the system state at the previous time step. More generally, for any set A of indices less than t and set of indices B greater than t we have:

$$P(s_t | \{s_i\}_{i \in A \cup B}) = P(s_t | s_{\max(A)}, s_{\min(B)}) \tag{2}$$

which follows from the Markov property.

Another useful identity which also follows directly from the Markov property is:

$$P(s_{t-1}, s_{t+1}|s_t) = P(s_{t-1}|s_t)P(s_{t+1}|s_t) \tag{3}$$

Discrete Markov Models. A important example of Markov chains are discrete Markov models. Each state s_t can take on one of a discrete set of states, and the probability of transitioning from one state to another is governed by a probability table for the whole sequence of states. More concretely, $s_t \in \{1, ..., K\}$ for some finite K and, for all times t, $P(s_t = j|s_{t-1} = i) = A_{ij}$ where A is parameter of the model that is a fixed matrix of valid probabilities (so that $A_{ij} \geq 0$ and $\sum_{j=1}^{K} A_{ij} = 1$). To fully characterize the model, we also require a distribution over states for the first time-step: $P(s_1 = i) = a_i$.

16.2 Hidden Markov Models

A Hidden Markov model (HMM) models a time-series of observations $\mathbf{y}_{1:T}$ as being determined by a "hidden" discrete Markov chain $s_{1:T}$. In particular, the measurement $\mathbf{y}_t$ is assumed to be determined by an "emission" distribution that depends on the hidden state at time t: $p(\mathbf{y}_t|s_t = i)$. The Markov chain is called "hidden" because we do not measure it, but must reason about it indirectly. Typically, s_t encodes underlying structure of the time-series, where as the $\mathbf{y}_t$ correspond to the measurements that are actually observed. For example, in speech modeling applications, the measurements $\mathbf{y}$ might be the waveforms measured from a microphone, and the hidden states might be the corresponding word that the speaker is uttering. In language modeling, the measurements might be discrete words, and the hidden states their underlying parts of speech.

HMMs can be used for discrete or continuous data; in this course, we will focus solely on the continuous case, with Gaussian emission distributions.

The joint distribution over observed and hidden is:

$$p(s_{1:T}, \mathbf{y}_{1:T}) = p(\mathbf{y}_{1:T}|s_{1:T})P(s_{1:T}) \tag{4}$$

where

$$P(s_{1:T}) = P(s_1)\prod_{t=2}^{T} P(s_t|s_{t-1}) \tag{5}$$

and

$$P(\mathbf{y}_{1:T}|s_{1:T}) = \prod_{t=1}^{T} p(\mathbf{y}_t|s_t) \tag{6}$$

The Gaussian model says:

$$p(\mathbf{y}_t|s_t = i) = \mathcal{N}(\mathbf{y}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{7}$$

for some mean and covariance parameters $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$. In other words, each state i has its own Gaussian with its own parameters. A complete HMM consists of the following parameters: a, A, $\boldsymbol{\mu}_{1:K}$,

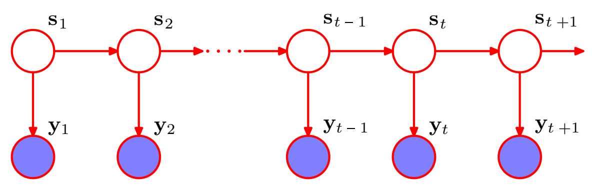

Figure 2: Illustration of the variables in an HMM, and their conditional dependencies. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

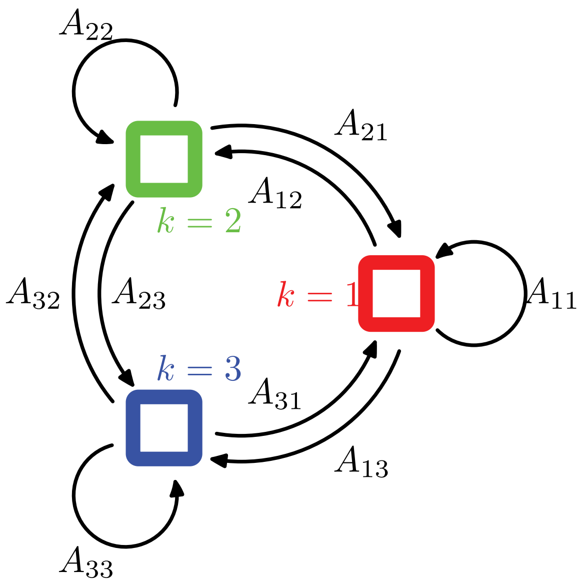

Figure 3: The hidden states of an HMM correspond to a state machine. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

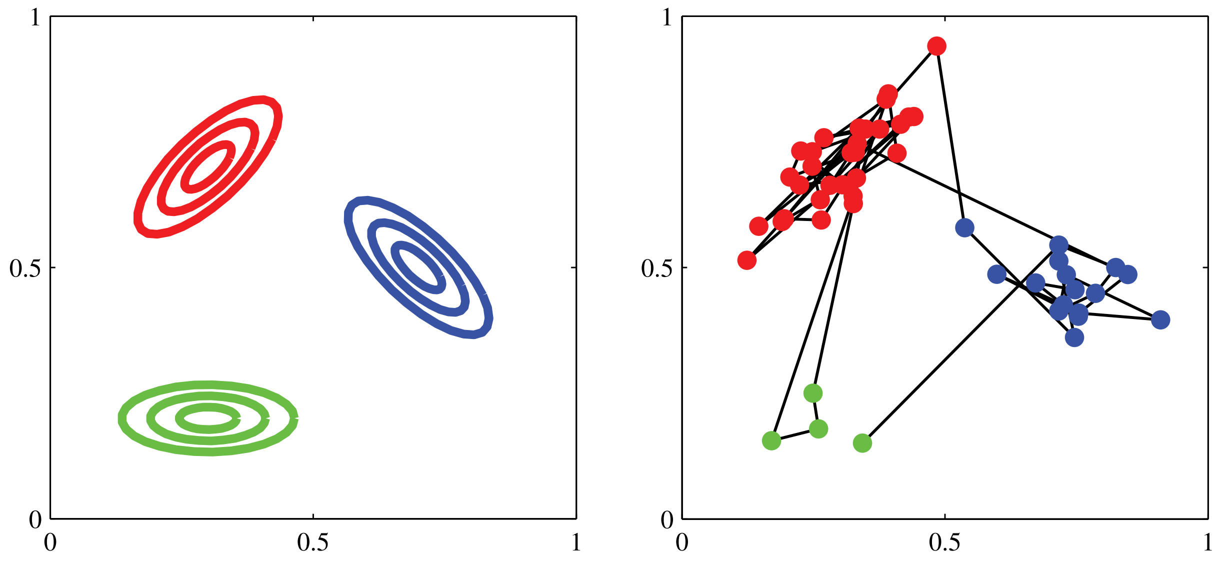

Figure 4: Illustration of sampling a sequence of datapoints from an HMM. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

and $\mathbf{\Sigma}_{1:K}$. As a short-hand, we will denote these parameters by a variable $\theta = \{a, A, \boldsymbol{\mu}_{1:K}, \mathbf{\Sigma}_{1:K}\}$.

Note that, if $A_{ij} = a_j$ for all i, then this model is equivalent to a Mixtures-of-Gaussian model with mixing proportions given by the a_i's, since the distribution over states at any instant does not depend on the previous state.

In the remainder of this chapter, we will discuss algorithms for computing with HMMs.

16.3 Viterbi Algorithm

We begin by considering the problem of computing the most-likely sequence of states given a data set $\mathbf{y}_{1:T}$ and a known HMM model. That is, we wish to compute

$$s^*_{1:T} = \arg\max_{s_{1:T}} P(s_{1:T}|\theta, \mathbf{y}_{1:T}) \tag{8}$$

The naive approach is to simply enumerate every possible state sequence and choose the one that maximizes the above conditional probability. Since there are K^T possible state-sequences, this approach is clearly infeasible for sequences of more than a few steps.

Fortunately, we can take advantage of the Markov property to perform this computation much more efficiently. The Viterbi algorithm is a dynamic programming approach to finding the most likely sequence of states $s_{1:T}$ given θ and a sequence of observations $\mathbf{y}_{1:T}$.

We begin by defining the following quantity for each state and each time-step:

$$\delta_t(i) \equiv \max_{s_{1:t-1}} p(s_{1:t-1}, s_t = i, \mathbf{y}_{1:t}) \tag{9}$$

(Henceforth, we omit θ from these equations for brevity.) This quantity tells us the likelihood that the most-likely sequence up to time t ends at state i, given the data up to time t. We will compute this quantity recursively. The base case is simply:

$$\delta_1(i) = p(s_1 = i, \mathbf{y}_1) = p(\mathbf{y}_1 | s_1 = i) P(s_1 = i) \tag{10}$$

for all i. The recursive case is:

$$\begin{aligned}
\delta_t(i) &= \max_{s_{1:t-1}} p(s_{1:t-1}, s_t = i, \mathbf{y}_{1:t}) \\
&= \max_{s_{1:t-2}, j} p(s_t = i | s_{t-1} = j) p(\mathbf{y}_t | s_t = i) p(s_{1:t-2}, s_{t-1} = j, \mathbf{y}_{1:t-1}) \\
&= p(\mathbf{y}_t | s_t = i) \max_j \left[p(s_t = i | s_{t-1} = j) \max_{s_{1:t-2}} p(s_{1:t-2}, s_{t-1} = j, \mathbf{y}_{1:t-1}) \right] \\
&= p(\mathbf{y}_t | s_t = i) \max_j A_{ji} \delta_{t-1}(j)
\end{aligned}$$

Once we have computed δ for all time-steps and all states, we can determine the final state of the most-likely sequence as:

$$\begin{aligned}
s_T^* &= \arg\max_i P(s_T = i | \mathbf{y}_{1:T}) && (11) \\
&= \arg\max_i P(s_T = i, \mathbf{y}_{1:T}) && (12) \\
&= \arg\max_i \delta_T(i) && (13)
\end{aligned}$$

since $p(\mathbf{y}_{1:T})$ does not depend on the state sequence. We can then backtrack through δ to determine the states of each previous time-step, by finding which state j was used to compute each maximum in the recursive step above. These states would normally be stored during the recursive process so that they can be looked-up later.

16.4 The Forward-Backward Algorithm

We may be interested in computing quantities such as $p(\mathbf{y}_{1:T} | \theta)$ or $P(s_t | \mathbf{y}_{1:T}, \theta)$; these distributions are useful for learning and analyzing models. Again, the naive approach to computing these quantities involves summing over all possible sequences of hidden states, and thus is intractable to compute. The Forward-Backward Algorithm allows us to compute these quantities in polynomial time, using dynamic programming.

In the **Forward Recursion**, we compute:

$$\alpha_t(i) \equiv p(\mathbf{y}_{1:t}, s_t = i) \tag{14}$$

The base case is:

$$\alpha_1(i) = p(\mathbf{y}_1 | s_1 = i) p(s_1 = i) \tag{15}$$

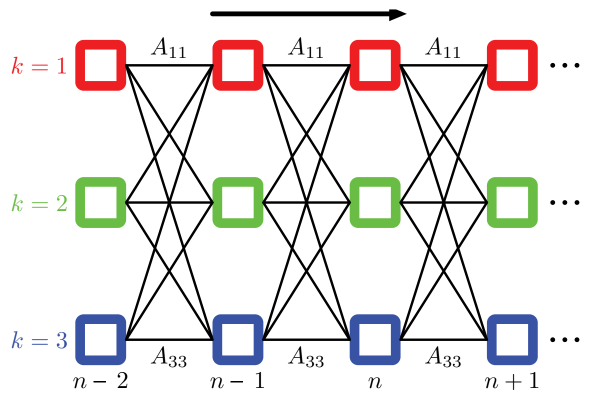

Figure 5: Illustration of the $\alpha_t(i)$ values computed during the Forward Recursion. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

and the recursive case is:

$$\alpha_t(i) = \sum_j p(\mathbf{y}_{1:t}, s_t = i, s_{t-1} = j) \tag{16}$$

$$= \sum_j p(\mathbf{y}_t | s_t = i) P(s_t = i | s_{t-1} = j) p(\mathbf{y}_{1:t-1}, s_{t-1} = j) \tag{17}$$

$$= p(\mathbf{y}_t | s_t = i) \sum_{j=1}^{K} A_{ji} \alpha_{t-1}(j) \tag{18}$$

Note that this is identical to the Viterbi algorithm, except that maximization over j has been replaced by summation.

In the **Backward Recursion** we compute:

$$\beta_t(i) \equiv p(\mathbf{y}_{t+1:T} | s_t = i) \tag{19}$$

The base case is:

$$\beta_T(i) = 1 \tag{20}$$

The recursive case is:

$$\beta_t(i) = \sum_{j=1}^{K} A_{ij} p(\mathbf{y}_{t+1} | s_{t+1} = j) \beta_{t+1}(j) \tag{21}$$

From these quantities, we can easily the following useful quantities.

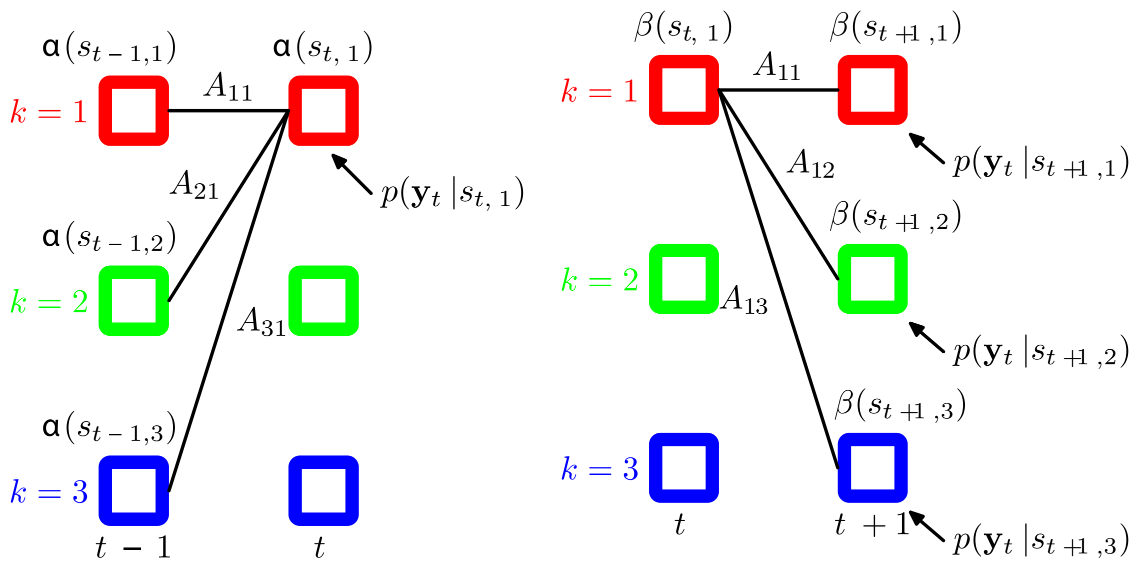

Figure 6: Illustration of the steps of the Forward Recursion and the Backward Recursion (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

The probability that the hidden sequence had state i at time t is:

$$\begin{aligned}
\gamma_t(i) &\equiv p(s_t = i|\mathbf{y}_{1:T}) && (22)\\
&= \frac{p(\mathbf{y}_{1:T}|s_t = i)p(s_t = i)}{p(\mathbf{y}_{1:T})} && (23)\\
&= \frac{p(\mathbf{y}_{1:t}|s_t = i)p(\mathbf{y}_{t+1:T}|s_t = i)p(s_t = i)}{p(\mathbf{y}_{1:T})} && (24)\\
&= \frac{\alpha_t(i)\beta_t(i)}{p(\mathbf{y}_{1:T})} && (25)
\end{aligned}$$

The normalizing constant — which is also the likelihood of the entire sequence, $p(\mathbf{y}_{1:T})$ — can be computed by the following formula:

$$\begin{aligned}
p(\mathbf{y}_{1:T}) &= \sum_i p(s_t = i, \mathbf{y}_{1:T}) && (26)\\
&= \sum_i \alpha_t(i)\beta_t(i) && (27)
\end{aligned}$$

The result of this summation will be the same regardless of which time-step t we choose to do the summation over.

The probability that the hidden sequence transitioned from state i at time t to state j at time

$t + 1$ is:

$$\xi_t(i,j) \equiv P(s_t = i, s_{t+1} = j | \mathbf{y}_{1:T}) \tag{28}$$
$$= \frac{\alpha_t(i) A_{ij} p(\mathbf{y}_{t+1} | s_{t+1} = j) \beta_{t+1}(j)}{p(\mathbf{y}_{1:T})} \tag{29}$$
$$= \frac{\alpha_t(i) A_{ij} p(\mathbf{y}_{t+1} | s_{t+1} = j) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i) A_{ij} p(\mathbf{y}_{t+1} | s_{t+1} = j) \beta_{t+1}(j)} \tag{30}$$

Note that the denominator gives an expression for $p(\mathbf{y}_{1:T})$, which can be computed for any value of t.

16.5 EM: The Baum-Welch Algorithm

Learning in HMMs is normally done by maximum likelihood, i.e., we wish to find the model parameters such that:

$$\theta^* = \arg\max_\theta p(\mathbf{y}_{1:T} | \theta) \tag{31}$$

As before, even evaluating this objective function requires K^T steps, and methods like gradient descent will be impractical. Instead, we can use the EM algorithm. Note that, since an HMM is a generalization of a Mixture-of-Gaussians, EM for HMMs will be a generalization of EM for MoGs. The EM algorithm applied to HMMs is also known as the Baum-Welch Algorithm.

The algorithm alternates between the following two steps:

- **The E-Step:** The Forward-Backward Algorithm is performed, in order to compute γ and ξ.
- **The M-Step:** The parameters θ are updated as follows:

$$a_i = \gamma_1(i) \tag{32}$$
$$\boldsymbol{\mu}_i = \frac{\sum_t \gamma_t(i) \mathbf{y}_t}{\sum_t \gamma_t(i)} \tag{33}$$
$$\boldsymbol{\Sigma}_i = \frac{\sum_t \gamma_t(i) (\mathbf{y}_t - \boldsymbol{\mu}_i)(\mathbf{y}_t - \boldsymbol{\mu}_i)^T}{\sum_t \gamma_t(i)} \tag{34}$$
$$A_{ij} = \frac{\sum_t \xi_t(i,j)}{\sum_k \sum_t \xi_t(i,k)} \tag{35}$$

16.5.1 Numerical issues: renormalization

In practice, numerical issues are a problem for the straightforward implementation of the Forward-Backward Algorithm. Since the α_t's involve joint probabilities over the entire sequence up to time t, they will be very small. In fact, as t grows, the values of α_t tend to shrink exponentially towards 0. Thus the limit of machine precision will quickly be reached and the computed values will underflow (evaluate to zero).

The solution is to compute a normalized terms in the Forward-Backward recursions:

$$\hat{\alpha}_t(i) = \frac{\alpha_i(t)}{\prod_{m=1}^{T} c_m} \tag{36}$$

$$\hat{\beta}_t(i) = \left(\prod_{m=t+1}^{T} c_m \right) \beta_i(t) \tag{37}$$

Specifically, we use $c_t = p(\mathbf{y}_t | \mathbf{y}_{1:t-1})$. It can then be seen that, if we use $\hat{\alpha}$ and $\hat{\beta}$ in the M-step instead of α and β, the c_t terms will cancel out (you can see this by substituting the formulas for γ and ξ into the M-step). We then must choose c_t to keep the scaling of $\hat{\alpha}$ and $\hat{\beta}$ within machine precision.

In the base case for the forward recursion, we set:

$$c_1 = \sum_{i=1}^{K} p(\mathbf{y}_1 | s_1 = i) a_i \tag{38}$$

$$\hat{\alpha}_1(i) = \frac{p(\mathbf{y}_1 | s_1 = i) a_i}{c_1} \tag{39}$$

(This may be implemented by first computing the numerator of $\hat{\alpha}$, and then summing it to get c_1). The recursion for computing $\hat{\alpha}$ is

$$c_t = \sum_i p(\mathbf{y}_t | s_t = i) \sum_{j=1}^{K} A_{ji} \hat{\alpha}_{t-1}(j) \tag{40}$$

$$\hat{\alpha}_t(i) = p(\mathbf{y}_t | s_t = i) \frac{\sum_{j=1}^{K} A_{ji} \hat{\alpha}_{t-1}(j)}{c_t} \tag{41}$$

In the backward step, the base case is:

$$\hat{\beta}_T(i) = 1 \tag{42}$$

and the recursive case is

$$\hat{\beta}_t(i) = \frac{\sum_{j=1}^{K} A_{ij} p(\mathbf{y}_{t+1} | s_{t+1} = j) \hat{\beta}_{t+1}(j)}{c_{t+1}} \tag{43}$$

using the same c_t values computed in the forward recursion.

The γ and ξ variables can then be computed as

$$\gamma_t(i) = \hat{\alpha}_t(i) \hat{\beta}_t(i) \tag{44}$$

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i) p(\mathbf{y}_{t+1} | s_{t+1} = j) A_{ij} \hat{\beta}_{t+1}(j)}{c_{t+1}} \tag{45}$$

It can be shown that $c_t = p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$. Hence, once the recursion is complete, we can compute the data likelihood as

$$p(\mathbf{y}_{1:T}) = \prod_t c_t \tag{46}$$

or, in the log-domain (which is more stable),

$$\ln p(\mathbf{y}_{1:T}) = \sum \ln c_t \tag{47}$$

This quantity is decreased after ever EM-step until convergence of EM.

16.5.2 Free Energy

EM can be viewed as optimizing the model parameters θ together with the distribution ξ.

The Free Energy for a Hidden Markov Model is:

$$\begin{aligned} F(\theta, \xi) &= -\sum_i \gamma_1(i) \ln a_i - \sum_{i,j} \sum_{t=1}^{T-1} \xi_t(i,j) \ln A_{ij} - \sum_i \sum_{t=1}^{T} \gamma_t(i) \ln p(\mathbf{y}_t|s_t = i) \\ &\quad + \sum_{i,j} \sum_{t=1}^{T-1} \xi_t(i,j) \ln \xi_t(i,j) - \sum_i \sum_{t=2}^{T-2} \gamma_t(i) \ln \gamma_t(i) \end{aligned} \tag{48}$$

where γ is defined as a function of ξ as:

$$\gamma_t(i) = \sum_k \xi_t(i,k) = \sum_k \xi_{t-1}(k,i) \tag{49}$$

Warning! Since we weren't able to find any formula for the free energy, we derived it from scratch (see below). In our tests, it didn't precisely match the negative log-likelihood. So there might be a mistake here, although the free energy did decrease as expected.

Derivation. This material is very advanced and not required for the course. It is mainly here because we couldn't find it elsewhere.

As a short-hand, we define $\mathbf{s} = s_{1:T}$ to be a variable representing an entire state sequence. The likelihood of a data sequence is:

$$p(\mathbf{y}_{1:T}) = \sum_{\mathbf{s}} p(\mathbf{y}_{1:T}, \mathbf{s}) \tag{50}$$

where the summation is over all possible state sequences.

In EM, we're really optimizing θ and a distribution $q(\mathbf{s})$ over the possible state sequences. The variable ξ is just one way of representing this distribution by its marginals; the variable γ are the marginals of ξ:

$$\begin{aligned}
\gamma_t(i) &= q(s_t = i) = \sum_{\mathbf{s}\backslash\{i\}} q(\mathbf{s}) && (51)\\
\xi_t(i,j) &= q(s_t = i, s_{t+1} = j) = \sum_{\mathbf{s}\backslash\{i,j\}} q(\mathbf{s}) && (52)
\end{aligned}$$

We can also compute the full distribution from ξ and γ:

$$\begin{aligned}
q(\mathbf{s}) &= q(s_1) \prod_{t=1}^{T-1} q(s_{t+1}|s_t) && (53)\\
&= \gamma_1(i) \prod_{t=1}^{T-1} \frac{\xi_t(i,j)}{\gamma_t(i)} && (54)\\
&= \frac{\prod_{t=1}^{T-1} \xi_t(i,j)}{\prod_{t=2}^{T-1} \gamma_t(i)} && (55)
\end{aligned}$$

The Free Energy is then:

$$\begin{aligned}
F(\theta, q) &= -\sum_{\mathbf{s}} q(\mathbf{s}) \ln p(\mathbf{s}, \mathbf{y}_{1:T}) + \sum_{\mathbf{s}} q(\mathbf{s}) \ln q(\mathbf{s}) && (56)\\
&= F_1(q) + F_2(q) && (57)
\end{aligned}$$

The first term can be decomposed as:

$$\begin{aligned}
F_1(q) &= -\sum_{\mathbf{s}} q(\mathbf{s}) \ln p(\mathbf{s}, \mathbf{y}_{1:T}) && (58)\\
&= \sum_{\mathbf{s}} q(\mathbf{s}) \ln \left(P(s_1) \prod_{t=1}^{T-1} P(s_{t+1}|s_t) \prod_{t=1}^{T} p(\mathbf{y}_t|s_t) \right) && (59)\\
&= -\sum_{\mathbf{s}} q(\mathbf{s}) \ln P(s_1) - \sum_{\mathbf{s}} \sum_t q(\mathbf{s}) \ln P(s_{t+1}|s_t) - \sum_{\mathbf{s}} \sum_t q(\mathbf{s}) \ln p(\mathbf{y}_t|s_t) && (60)\\
&= -\sum_i \gamma_1(i) \ln P(s_1 = i) - \sum_{i,j,t} \xi_t(i,j) \ln P(s_{t+1} = j | s_t = i)\\
&\quad - \sum_{i,t} \gamma_t(i) \ln p(\mathbf{y}_t | s_t = i) && (61)
\end{aligned}$$

The second term can be simplified as:

$$
\begin{aligned}
F_2(q) &= \sum_{\mathbf{s}} q(\mathbf{s}) \ln q(\mathbf{s}) && (62)\\
&= \sum_{\mathbf{s}} q(\mathbf{s}) \ln \frac{\prod_{t=1}^{T-1} \xi_t(i,j)}{\prod_{t=2}^{T-1} \gamma_t(i)} && (63)\\
&= \sum_{\mathbf{s}} \sum_{t=1}^{T-1} q(\mathbf{s}) \ln \xi_t(i,j) - \sum_{\mathbf{s}} \sum_{t=2}^{T-1} q(\mathbf{s}) \ln \gamma_t(i) && (64)\\
&= \sum_{i,j} \sum_{t=1}^{T-1} \xi_t(i,j) \ln \xi_t(i,j) - \sum_{i} \sum_{t=2}^{T-1} \gamma_t(i) \ln \gamma_t(i) && (65)
\end{aligned}
$$

16.6 Most likely state sequences

Suppose we wanted to computed the most likely states s_t for each time in a sequence. There are two ways that we might do it: we could take the **most likely state sequence**:

$$s^*_{1:T} = \arg\max_{s_{1:T}} p(s_{1:T}|\mathbf{y}_{1:T}) \tag{66}$$

or we could take **the sequence of most-likely states**:

$$s^*_t = \arg\max_{s_t} p(s_t|\mathbf{y}_{1:T}) \tag{67}$$

While these sequences may often be similar, they can be different as well. For example, it is possible that the most likely states for two consecutive time-steps do not have a valid transition between them, i.e., if $s^*_t = i$ and $s^*_{t+1} = j$, it is possible (though unlikely) that $A_{ij} = 0$. This illustrates that these two ways to create sequences of states answer two different questions: what sequence is jointly most likely? And, for each time-step, what is the most likely state just for that time-step?