1 Mixture models

The simplest interesting graphical model is the mixture model, with a state variable $Q$ and an output variable $Y$ governed by $P_{QY} = P_Q P_{Y|Q}$:

This can be viewed as a generative model that generates samples independently of each other by the following process:

1. for sample $i$, choose state $q_i$ according to $P_Q$;
2. then choose output $y_i$ according to $P_{Y|q_i}$.

In typical applications the state $q_i$ is hidden (aka latent), meaning that it is not directly observable but can only be inferred from $y_i$.

If $Q = \{1 \ldots n\}$, $Y = \mathbb{R}^k$, and $P(y|q) \sim \mathcal{N}(\mu_q, \Sigma_q)$ then $P_Y$ is an $n$-component Gaussian mixture with mixture weights $P(q)$, with pdf
\[
P(y) = \sum_{q=1}^{n} P(q) \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp\left(-\frac{1}{2}(y - \mu_q)'\Sigma_q^{-1}(y - \mu_q)\right).
\]

For example, an equally weighted 3-component mixture in \(\mathbb{R}^2\):

Mixture models are often used for classification tasks. In training, labeled pairs \((q, y)\) are used to estimate the distributions \(P(q)\) and \(P(y|q)\). Then given an unlabeled point \(y\) one can calculate a distribution over the hidden label (state) using Bayes’ rule:

\[
P(q|y) = \frac{P(y, q)}{P(y)} \propto P(y|q)P(q).
\]

For example, one might use a mixture model to classify small \(n \times n\) binary images of handwritten characters into labels A..Z0..9. Although far from optimal, a simple parameterization is \(P(y = y_1 \ldots y_n|q) = \prod_{ij} P_{ijq}(y_{ij})\); this naive Bayes model is equivalent to the graphical model:

\[\text{\footnotesize{This is playing fast and loose with notation: for discrete } Y, P(y, q) \text{ and } P(y) \text{ are probabilities; for continuous } Y, P(y, q) \text{ and } P(y) \text{ are probability densities, but the formula still holds.}}\]
Why is this model “naive”?

2 Hidden Markov models

The hidden Markov model (HMM) is a generalization of the mixture model that allows for dependence between the state of samples. It is typically used when samples have a natural order in time or space. The generative process is

1. for sample 1, choose state $q_1$ according to $P_{Q_1}$; for subsequent samples, choose $q_i$ according to $P_{Q_i|q_{i-1}}$;
2. then choose output $y_i$ according to $P_{Y_i|q_i}$.

Vitally, the probability of state $q_i$ is dependent on $q_{i-1}$ but not previous states: $\forall i < j < k, Q_i \perp Q_k|Q_j$. 

3
In the example of handwritten character recognition, the hidden state dependence can be used to model structure in character sequences, for example the tendency for vowels to follow consonants and vice versa.

Consider again the 3-component Gaussian mixture assuming at every step the state progression is governed by

\[ P(q_{i+1}|q_i) = \begin{cases} .8 & \text{if } q_{i+1} = q_i \\ .1 & \text{otherwise} \end{cases} \]

In that case, the marginal probability distribution is the same as before (all states are equally likely) but consecutive samples are highly correlated:

Using knowledge of the state dynamics and output history \( y_1 \ldots y_i \), it is possible to dramatically improve our predictive power. We will see how to calculate \( P(q_{i+1}|y_1 \ldots y_i) \), and from that \( P(y_{i+1}|y_1 \ldots y_i) \).
3 Operations

HMMs are used many different ways:

- **Smoothing**: If $Q$ represents a hidden true signal that is corrupted by independent noise to produce an observed signal $Y$, then the distribution $P_{Q|y}$ can be used to “de-noise”, or smooth, the observation. A simple model of noise is $P_{Y|q} \sim \mathcal{N}(q, \sigma)$.

- **Prediction**: Given a sequence of observations $y_1 \ldots y_t$, predict $y_{t+1}$ using the identity $P(y_{t+1}|y_1 \ldots y_t) = \sum_{q_{t+1}} P(q_{t+1}|y_1 \ldots y_t)P(y_{t+1}|q_{t+1})$. Such models are commonly used in target tracking, where $Q$ represents a true position and $Y$ noisy measurements of it. In some text entry devices predictive probabilities are used to alter the size or order of keys to improve entry speed and accuracy.

- **Classification**: $Y$ represents a sequential input and $Q$ a desired classification or labeling of its parts. For example, in an information retrieval application $y_t$ may be a word from a document $y_1 \ldots y_n$ and $q_t$ a classification of the word into one of NameStart, NameMiddle, NameEnd, Other. Model estimation might take place from labeled examples; extraction of names from documents may be performed by calculating $\arg \max_q P(q|y)$. 

\[ P(y_{t+1}) \]  
\[ P(y_{t+1}|y_1 \ldots t) \]  
(last 5 samples shown)
4 Conditional Independence in HMMs

Using the chain rule,

\[
P(q, y) = P(q_1 \ldots q_n, y_1 \ldots y_n) = P(q_1)P(q_2|q_1)P(q_3|q_1q_2) \ldots P(q_n|q_1 \ldots q_{n-1}) \\
P(y_1|q_1 \ldots q_n)P(y_2|q_1 \ldots q_n, y_1) \ldots P(y_n|q_1 \ldots q_n, y_1 \ldots y_{n-1}).
\]

The HMM graphical model implies the conditional independencies:

\[
P(q_i|q_1 \ldots q_{i-1}) = P(q_i|q_{i-1}) \\
P(y_i|q_1 \ldots q_n, y_1 \ldots y_{i-1}) = P(y_i|q_i).
\]

Therefore

\[
P(q, y) = P(q_1) \prod_{i=2 \ldots n} P(q_i|q_{i-1}) \prod_{i=1 \ldots n} P(y_i|q_i).
\]

Rewriting pairwise conditional probabilities into pairwise joint probabilities, notice
\[
P(q, y) = P(q_1) \prod_{i=2 \ldots n} \frac{P(q_{i-1}, q_i)}{P(q_{i-1})} \prod_{i=1 \ldots n} \frac{P(q_i, y_i)}{P(q_i)}
\]
\[
= \frac{\prod_{i=2 \ldots n} P(q_{i-1}, q_i) \prod_{i=1 \ldots n} P(q_i, y_i)}{\prod_{i=1 \ldots n} P(q_i)^{\delta(Q_i)-1}}
\]

where \(\delta(Q_i)\) is the degree of variable \(Q_i\) in the graphical model. Simply relabeling the terms gives

\[
P(q, y) = \frac{\prod_{i=2 \ldots n} \Phi_{Q_i-1, Q_i}(q_{i-1}, q_i) \prod_{i=1 \ldots n} \Phi_{Q_i, Y_i}(q_i, y_i)}{\prod_{i=1 \ldots n} \Phi_{Q_i}(q_i)^{\delta(Q_i)-1}}
\]

an expression that shows directed HMMs are equivalent to undirected graphical models. Thus, all of the following can express the same distributions:
In fact, we can generalize the expression of the joint probability one step further to

\[
P(x) = \frac{\prod_{(s,t) \in E} \Phi_{st}(x_s, x_t)}{\prod_{v \in V} \Phi_v(x_v)^{\delta(X_v) - 1}}
\]

where \( V \) are the vertices in the graphical model and \( E \) are the edges. This expression holds for any acyclic undirected graphical model (tree model), and directed graphical models where no node has more than one parent.

5 Inference

Let us calculate \( P(y) = \sum_q P(q, y) \), a computation that at first blush appears to require a summation over an exponential (in \( n \)) number of hidden state configurations.

\[
P(y) = \sum_q P(q, y) = \sum_{q_1...q_n} P(q_1) \prod_{i=2...n} P(q_i|q_{i-1}) \prod_{i=1...n} P(y_i|q_i)
\]

\[
= \sum_{q_1} P(q_1)P(y_1|q_1) \sum_{q_2...q_n} \prod_{i=2...n} P(q_i|q_{i-1})P(y_i|q_i)
\]

\[
= \sum_{q_1} P(q_1)P(y_1|q_1) \sum_{q_2} P(q_2|q_1)P(y_2|q_2) \sum_{q_3...q_n} \prod_{i=3...n} P(q_i|q_{i-1})P(y_i|q_i)
\]

\[
= \vdots
\]

\[
= \sum_{q_1} P(q_1)P(y_1|q_1) \sum_{q_2} P(q_2|q_1)P(y_2|q_2) \sum_{q_3} \cdots \sum_{q_n} P(q_n|q_{n-1})P(y_n|q_n).
\]

This re-expression requires that multiplication (left-)distributes over addition: \( cx + cy = c(x+y) \); there are many operator combinations \( \otimes \) and \( \oplus \) with this property, besides ordinary
multiplication and addition of reals. The algorithms developed below can be generalized by choosing other operators.

It’s clear that the key terms of the computation are the products $P(q_i|q_{i-1})P(y_i|q_i)$, so to simplify notation let

$$A^{(1)}(q_i) \equiv P(q_i, y_i) = P(q_i)P(y_i|q_i)$$

$$A^{(i)}(q_{i-1}, q_i) \equiv P(q_i, y_i|q_{i-1}) = P(q_i|q_{i-1})P(y_i|q_i) \text{ for } 2 \leq i \leq n$$

Notice that $A$ is a function of the observation $y$! Then,

$$P(y) = \sum_{q_1} A^{(1)}(q_1) \sum_{q_2} A^{(2)}(q_1, q_2) \sum_{q_3} \ldots \sum_{q_n} A^{(n)}(q_{n-1}, q_n).$$

For any $i$ the inner expression $\sum_{q_{i+1}} A^{(i+1)}(q_i, q_{i+1}) \sum_{q_{i+2}} \ldots$ depends on $q_i$ but, thanks to the conditional independence assumption of Markov models, no state $q_j$ for $j < i$. Hence we can write:

$$P(y) = \sum_{q_1} A^{(1)}(q_1) \ldots \sum_{q_i} A^{(i)}(q_{i-1}, q_i) \sum_{q_{i+1}} \ldots \sum_{q_{i+1}} A^{(i+1)}(q_i, q_{i+1}) \ldots \sum_{q_n} A^{(n)}(q_{n-1}, q_n)$$

$$= \sum_{q_1} A^{(1)}(q_1) \ldots \sum_{q_i} A^{(i)}(q_{i-1}, q_i) \beta^{(i)}(q_i).$$

Computationally, we can calculate the backward probability vector $\beta^{(i)}$ once and store it, rather than calculating it once per configuration of $q_1 \ldots q_{i-1}$.
The conditioning state \( q_i \) can be further elevated by moving its summation to the front and capturing the remaining summations over \( q_1 \ldots q_{i-1} \) as a forward probability \( \alpha^{(i)} \):

\[
P(y) = \sum_{q_i} \left( \sum_{q_1} \prod_{i=2}^{i-1} A(q_{i-2}, q_{i-1}) A(q_{i-1}, q_i) \right) \beta^{(i)}(q_i) \\
= \sum_{q_i} \alpha^{(i)}(q_i) \beta^{(i)}(q_i) \\
\equiv \sum_{q_i} P(y_1 \ldots y_i, q_i) P(y_{i+1} \ldots y_n | q_i) \\
= \sum_{q_i} P(y_1 \ldots y_n, q_i).
\]

Here we see the power of conditional independence: choosing \( i = \frac{n}{2} \), the calculation of \( P(y) \) is reduced to two problems of half the size. Thus, instead of a sum of \(|Q|^n\) terms, 2 sums of \(|Q|^{\frac{n}{2}}\) terms. In fact, \( \alpha^{(i)} \) and \( \beta^{(i)} \) can both be computed recursively:

\[
\alpha^{(i)}(q_i) = \sum_{q_1} \prod_{i=2}^{i-1} A(q_{i-2}, q_{i-1}) A(q_{i-1}, q_i) \\
= \sum_{q_{i-1}} \alpha^{(i-1)}(q_{i-1}) A(q_{i-1}, q_i)
\]

where \( \alpha^{(1)}(q_1) = A^{(1)}(q_1) \), and
\[
\beta^{(i)}(q_i) = \sum_{q_{i+1}} A^{(i+1)}(q_i, q_{i+1}) \cdots \sum_{q_n} A^{(n)}(q_{n-1}, q_n) \\
= \sum_{q_{i+1}} A^{(i+1)}(q_i, q_{i+1}) \beta^{(i+1)}(q_{i+1})
\]

where \(\beta^{(n)}(q_n) = 1\).

5.1 Matrix formulation

Let us express the equations so far using matrix notation, treating \(\alpha^{(i)}(q_i)\) as a row vector \(\alpha^{(i)}\), \(\beta^{(i)}(q_i)\) as a column vector \(\beta^{(i)}\), and \(A^{(i)}(q_{i-1}, q_i)\) as a \(|Q_{i-1}| \times |Q_i|\) matrix \(A^{(i)}\). The matrix \(A^{(1)}\) is a \(1 \times |Q_1|\) row and the matrix \(A^{(n+1)}\) is a \(|Q_n| \times 1\) column. The forward probability calculation is a sequence of vector-matrix products and the backward probability calculation a sequence of matrix-vector products:

\[
A^{(n+1)} = 1 \\
\alpha^{(i)} = \begin{cases} 
A^{(1)} & \text{for } i = 1 \\
\alpha^{(i-1)} A^{(i)} & \text{for } i = 2 \ldots (n + 1) 
\end{cases} \\
\beta^{(i)} = \begin{cases} 
A^{(n+1)} & \text{for } i = n \\
A^{(i+1)} \beta^{(i+1)} & \text{for } i = (n - 1) \ldots 0 
\end{cases}
\]

The calculation of \(P(y)\) can be performed using a vector-vector product of forward and backward probabilities at any point \(i\):
\[ P(y) \equiv \sum_{q_i} P(y_1 \ldots y_i, q_i)P(y_{i+1} \ldots y_n|q_i) \]
\[ = \alpha^{(i)}\beta^{(i)} \text{ for any } 1 \leq i \leq n \]
\[ = A^{(1)}A^{(2)} \ldots A^{(i)}A^{(i+1)} \ldots A^{(n)}A^{(n+1)} \]
\[ = \alpha^{(n+1)} = \beta^{(0)}. \]

Finally, the marginal conditional probabilities \( \gamma(q_i) \equiv P(q_i|y) \), also known as the complete probabilities, can be calculated from forward and backward probabilities as a simple scalar product with normalization; this is the famous forward-backward algorithm:

\[ \gamma(q_i) \equiv \frac{P(q_i|y)}{P(y)} \]
\[ = \frac{P(y_1 \ldots y_i, q_i)P(y_{i+1} \ldots y_n|q_i)}{P(y)} \]
\[ = \frac{\alpha^{(i)}q_i\beta^{(i)}}{\beta^{(0)}}. \]

### 5.2 Trellis diagram

The trellis diagram below depicts the calculation of \( P(y) \) for an HMM with 4 steps and 3 possible state values \( q_i \in \{1, 2, 3\} \) at each step; \( P(y) = A^{(1)}A^{(2)}A^{(3)}A^{(4)}A^{(5)} \). Computing this product of matrices is equivalent to summing over all left-to-right paths through the trellis, each representing a particular hidden state sequence. The calculation is made efficient by summarizing intermediate results in forward probability vectors \( \alpha^{(i)} \).
A slight modification to the diagram shows how the forward-backward algorithm efficiently calculates $\gamma(q_i) = P(q_i|y) = \alpha^{(i)}_q \beta^{(i)}_q / P(y)$.

5.3 The forward-backward algorithm

Here, explicitly, is the forward-backward algorithm for computing the marginal posterior (complete) probabilities $\gamma(q_i) = P(q_i|y)$ for every $i, q_i$. It computes and stores forward and backward probabilities given input $y$ for every position and state value, and from them the complete probabilities, using $\Theta(n|Q|^2)$ time and $\Theta(n|Q|)$ space. More efficient algorithms, discussed further below, consume substantially less space.
$$(p, \gamma[]) = \text{FORWARD-BACKWARD}(y = y_1 \ldots y_n)$$

```plaintext
\[
\alpha[] = \text{new array}[n] \quad \text{// } [i][q], \text{ initialized to 0, forward probs}
\beta[] = \text{new array}[n] \quad \text{// } [i][q], \text{ initialized to 0, backward probs}
\gamma[] = \text{new array}[n] \quad \text{// } [i][q], \text{ initialized to 0, complete probs}
\]

// compute forward probabilities
for \( i = 1 \ldots n \)
  \( \alpha[i] = \text{new array}[|Q_i|] \)
  for \( q = 1 \ldots |Q_i| \)
    if \( i == 1 \)
      \( \alpha[i][q] = P_{Q_1}(q) \times P_{Y_i|Q_1}(y[i]|q) \)
    else
      for \( r = 1 \ldots |Q_{i-1}| \)
        \( \alpha[i][q] += \alpha[i-1][r] \times P_{Q_i|Q_{i-1}}(q|r) \times P_{Y_i|Q_i}(y[i]|q) \)

// compute backward and complete probabilities
for \( i = n \ldots 1 \)
  \( \beta[i] = \text{new array}[|Q_i|] \)
  \( \gamma[i] = \text{new array}[|Q_i|] \)
  for \( q = 1 \ldots |Q_i| \)
    if \( i == n \)
      \( \beta[i][q] = 1 \)
    else
      for \( r = 1 \ldots |Q_{i+1}| \)
        \( \beta[i][q] += \beta[i+1][r] \times P_{Q_i|Q_{i+1}}(r|q) \times P_{Y_{i+1}|Q_{i+1}}(y[i+1]|r) \)
      \( \gamma[i][q] = \alpha[i][q] \times \beta[i][q] / p \)

return \( p, \gamma[] \)
```
5.4 Associativity and order of operations

The matrix formulation of HMM operations lets us use our understanding of matrix multiplication to gain insight into computation in graphical models. For example, from

\[ P(y) = A^{(1)}A^{(2)} \ldots A^{(n)}A^{(n+1)} \]

it is clear that despite starting from a generative definition of HMMs that proceeds left-to-right, in fact by the associative nature of matrix multiplication the calculation can be performed in many orders:

\[
((((((A^{(1)}A^{(2)})A^{(3)})A^{(4)})A^{(5)})A^{(6)})A^{(7)})A^{(8)}) \quad \text{left to right}
\]
\[
((A^{(1)}A^{(2)})(A^{(3)}A^{(4)}))((A^{(5)}A^{(6)})(A^{(7)}A^{(8)})) \quad \text{right to left}
\]
\[
((A^{(1)})(A^{(2)})(A^{(3)})(A^{(4)})) \quad \text{divide-and-conquer}
\]

Certain orders may have better computational properties than others: the divide-and-conquer algorithm multiplies \(|Q| \times |Q|\) matrices, an operation that using standard algorithms on a sequential computer takes \(\Theta(n|Q|^3)\) total time, whereas the traditional left-to-right calculation (of forward probabilities \(\alpha\)) and right-to-left calculation (of backward probabilities \(\beta\)) summarize results in vectors and perform only matrix-vector products, take \(\Theta(n|Q|^2)\) time, a substantial savings for large \(|Q|\). However given the possibility of parallel computation, the divide-and-conquer strategy may be preferable, as the computation depth is reduced from \(\Theta(n)\) to \(\Theta(\log n)\).

5.5 Variable elimination

For HMMs, multivariate conditional distributions over hidden variables can be calculated efficiently by variable elimination, a general inference technique that computes marginal distributions over subsets of variables by constructing a modified graphical model containing only those variables.

Consider the marginal distribution \(P_{Q_iQ_jQ_k|y}\) of the hidden variables \(Q_i, Q_j, Q_k\) (where \(i < j < k\) are not necessarily consecutive), after conditioning on the observation \(Y = y\):
\[
P(q_i, q_j, q_k | y) \\
= \frac{P(y, q_i, q_j, q_k)}{P(y)} \\
= \frac{[A^{(1)} \ldots A^{(i)}]_{q_i} [A^{(i+1)} \ldots A^{(j)}]_{q_i, q_j} [A^{(j+1)} \ldots A^{(k)}]_{q_j, q_k} [A^{(k+1)} \ldots A^{(n+1)}]_{q_k}}{P(y)} \\
= \frac{\alpha^{(i)}_{q_i} A^{(i+1,j)}_{q_i, q_j} A^{(j+1,k)}_{q_j, q_k} \beta^{(k)}_{q_k}}{P(y)}.
\]

Thus, \( P_{Q_i Q_j Q_k | y} \) can be computed efficiently using tables over state pairs constructed by partitioning \( A^{(1)} \ldots A^{(n+1)} \). Furthermore, since

\[
P_{Q_i Q_j Q_k | y} = \Phi^y_{Q_i Q_j} (q_i, q_j) \Phi^y_{Q_j Q_k} (q_j, q_k)
\]

where

\[
\Phi^y_{Q_i Q_j} (q_i, q_j) = \frac{\alpha^{(i)}_{q_i} A^{(i+1,j)}_{q_i, q_j}}{P(y)}, \\
\Phi^y_{Q_j Q_k} (q_j, q_k) = A^{(j+1,k)}_{q_j, q_k} \beta^{(k)}_{q_k}.
\]

\( P_{Q_i Q_j Q_k | y} \) is an undirected graphical model of form:

\[
\begin{array}{ccc}
Q_i & - & Q_j \\
| & & | \\
Q_j & - & Q_k
\end{array}
\]
It is also easily verified that $P_{Q_i Q_j Q_k | y} = P_{Q_i | y} P_{Q_j | Q_i, y} P_{Q_k | Q_j, y}$ where

$$
P_{Q_i | y} = \frac{\alpha^{(i)}_{q_i} \beta^{(i)}_{q_i}}{P(y)}
$$

$$
P_{Q_j | Q_i, y} = \frac{P_{Q_j, Q_i | y}}{P_{Q_i | y}} = \frac{\alpha^{(i)}_{q_i} A^{(i+1, j)}_{q_i, q_j} \beta^{(j)}_{q_j}}{\alpha^{(i)}_{q_i} \beta^{(i)}_{q_i}} = \frac{A^{(i+1, j)}_{q_i, q_j} \beta^{(j)}_{q_j}}{\beta^{(i)}_{q_i}}
$$

$$
P_{Q_k | Q_j, y} = \frac{P_{Q_k, Q_j | y}}{P_{Q_j | y}} = \frac{\alpha^{(j)}_{q_j} A^{(j+1, k)}_{q_j, q_k} \beta^{(k)}_{q_k}}{\alpha^{(j)}_{q_j} \beta^{(j)}_{q_j}} = \frac{A^{(j+1, k)}_{q_j, q_k} \beta^{(k)}_{q_k}}{\beta^{(j)}_{q_j}}.
$$

consistent with the directed graphical model

\[ Q_i \rightarrow Q_j \rightarrow Q_k \]

It is not true for general graphical models that the marginal distribution over a subset of variables takes such a simple form; the hidden variables of HMMs (and more generally, tree graphical models) are a special case.

### 6 Variations

#### 6.1 The Viterbi algorithm

In smoothing, classification and transduction tasks one frequently desires the most probable configuration $q^*$ of the hidden variables given an observation:
\[ q^* = \arg \max_q P(q|y) \]
\[ = \arg \max_q P(q, y). \]

The configuration \( q^* \) is known as the Viterbi sequence. It can be computed efficiently by replacing the summation in the forward (or backward) calculation with maximization, so that instead of the usual

\[ \alpha_{q_i}^{(i)} \equiv P(y_1 \ldots y_i, q_i) = \sum_{q_1 \ldots q_{i-1}} P(y_1 \ldots y_i, q_1 \ldots q_{i-1} q_i) \]

the forward probabilities are instead defined

\[ \alpha_{q_i}^{(i)} \equiv \max_{q_1 \ldots q_{i-1}} P(y_1 \ldots y_i, q_1 \ldots q_{i-1} q_i). \]

In more detail, let the optimal prefix \( \pi \) for a state value \( q_i \) be the sequence \( q_1 \ldots q_i \) that maximizes the probability of \( y_1 \ldots y_i \). It simplifies bookkeeping to store optimal prefixes \( \pi_{q_i}^{(i)} \) and their probabilities together in a pair:

\[ \pi_{q_i}^{(i)} = \arg \max_{q_1 \ldots q_{i-1} q_i} P(y_1 \ldots y_i, q_1 \ldots q_{i-1} q_i) \]
\[ \alpha_{q_i}^{(i)} \equiv \langle P(y_1 \ldots y_i, \pi_{q_i}^{(i)}); \pi_{q_i}^{(i)} \rangle. \]

It remains possible to express the computation of \( \alpha_{q_i}^{(i)} \) using matrix notation if the \( \oplus \) operator maximize probabilities and the \( \otimes \) operator concatenates state sequences:
\[ A_{q_1}^{(1)} = \langle P(q_1)P(y_1|q_1) ; q_1 \rangle \]
\[ A_{q_{i-1}q_i}^{(2 \leq i \leq n)} = \langle P(q_i|q_{i-1})P(y_i|q_i) ; q_i \rangle \]
\[ A_{q_{n+1}}^{(n+1)} = \langle 1 ; \varepsilon \rangle \]
\[ \otimes(\langle p_1 ; s_1 \rangle, \langle p_2 ; s_2 \rangle) = \langle p_1 p_2 ; \text{concat}(s_1, s_2) \rangle \]
\[ \oplus(\langle p_1 ; s_1 \rangle, \langle p_2 ; s_2 \rangle) = \begin{cases} \langle p_1 ; s_1 \rangle & \text{if } p_1 \geq p_2 \\ \langle p_2 ; s_2 \rangle & \text{otherwise} \end{cases} \]

From this we have \[ \langle P(y, q^*) ; q^* \rangle = A^{(1)} A^{(2)} \ldots A^{(n+1)} \].

An implementation of the Viterbi algorithm:

\[ q^* = \text{VITERBI}(y = y_1 \ldots y_n) \]
\[ \alpha[] = \text{nil} // \text{probabilities for state } i - 1 \]
\[ \pi[] = \text{nil} // \text{optimal prefixes for state } i - 1 \]
for \( i = 1 \ldots n \)
\[ \alpha[] = \text{new array}[|Q_i|] // \text{probabilities for state } i \]
\[ \pi[] = \text{new array}[|Q_i|] // \text{optimal prefixes for state } i \]
for \( q = 1 \ldots |Q_i| \)
if \( i == 1 \)
\[ \alpha[q] = P_{Q_1}(q) * P_{Y_i|Q_i}(y[i]|q) \]
\[ \pi[q] = q \]
else
\[ \alpha[q] = -\infty \]
for \( r = 1 \ldots |Q_{i-1}| \)
\[ p = \alpha'[r] * P_{Q_i|Q_{i-1}}(q|r) * P_{Y_i|Q_i}(y[i]|q) \]
if \( p > \alpha[q] \)
\[ \alpha[q] = p \]
\[ \pi[q] = \text{concat}(\pi'[r], q) \]
\[ \alpha' = \alpha // \text{new becomes old} \]
\[ \pi' = \pi // \text{new becomes old} \]
// compute best of final states
p = -\infty
q^* = nil
for q = 1\ldots|Q_n|
    if \alpha'[q] > p
        p = \alpha'[q]
        q^* = \pi'[q]
return q^*

6.2 States and output

6.3 Variable horizon

7 Practicalities

7.1 Memory consumption

Both the forward-backward algorithm and the Viterbi algorithm require $\Theta(n|Q|)$ storage space if implemented naively.

7.2 Numerical underflow

7.3 Continuous distributions

7.4 Large state space