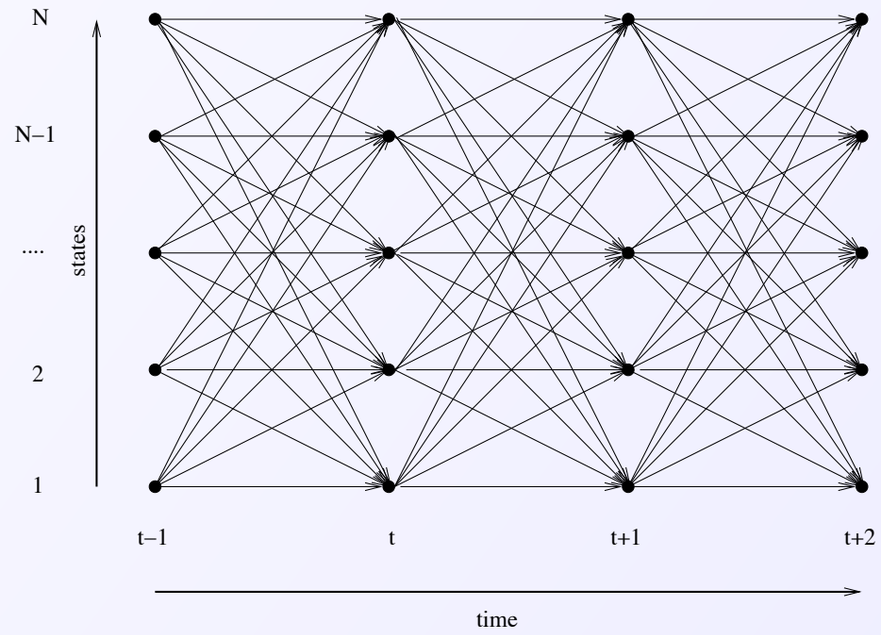
COM4511/COM6511 - Speech Technology

Lecture 7
Hidden Markov Models for Speech Recognition



Thomas Hain
t.hain@sheffield.ac.uk
Spring Semester

Outline

- Isolated word recognition
- How to use HMMs for speech recognition
- The Viterbi algorithm
- HMM training
- Extending Gaussians.
- (The Baum-Welch algorithm)

Isolated word recognition

If individual words are spoken in isolation we can reduce the complexity of the search for the best word sequence. We assume that words are separated from each other (in a sentence) by detectable pauses. The task for the acoustic model is simplified to providing an estimation of probabilities for words individually. The front-end produces a stream of feature vectors (**observation vectors**)

$$\mathbf{O} = [\mathbf{o}_1, \ldots, \mathbf{o}_t, \ldots, \mathbf{o}_T]$$

In continuous ASR we have to search for the best word sequence $\hat{W}$:

$$\hat{W} = \arg\max_W p(\mathbf{O}|W)P(W) \tag{1}$$

The audio stream can be separated by segmentation of the audio signal (end point detection) into M segments $\mathbf{O}^{(i)}$, where $i = 1 \ldots M$. Note that in that case the sentence length (the number of words) is directly derived from the audio signal (different to continuous speech recognition).

Isolated word recognition - Word classification

Each audio stream $\mathbf{O}^{(i)}$ is associated with one particular word out of a sentence $W = [w_1, w_2, \ldots, w_i, \ldots, w_M]$). The acoustic realisation of one word can be assumed to be independent of the neighbouring words. Thus we can simplify eq. (1) to give

Isolated word recognition

$$\hat{W} = \arg\max_W P(W) \prod_{i=1}^{M} p(\mathbf{O}^{(i)}|w_i)$$

If words are assumed to be completely independent we speak of **word classification**. Using a vocabulary of size N we can choose the index of the likely word k:

$$k = \arg\max_{1 \leq i \leq N} p(\mathbf{O}|w_i)P(w_i) \tag{2}$$

This allows a linear search for the best matching model. If all words are equally likely ($P(w_i) = \frac{1}{N}$): pick the word w with the **highest likelihood** $p(\mathbf{O}|w)$.

Isolated word recognition

Using Gaussian PDFs for the output distributions the HMM is defined by

- ➜ The transition probabilities a_{ij}
  These implicitly define the topology of the HMM.
- ➜ The mean vectors $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$
  Usually only diagonal covariance matrices are used (uncorrelated vector elements).

If we can compute the likelihood of an observation sequence we can use eq. (2) to select the most likely word. Models for each word in the vocabulary are needed. The model for a particular word is represented by the model parameters:

$$\lambda_w = \{\{a_{ij}\}, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$$

The N word models make up the parameter set of the acoustic model:

$$\lambda = \left\{\lambda_{w_1}, \lambda_{w_2}, \ldots, \lambda_{w_N}\right\}$$

Questions

The use of HMMs (for ASR) requires answers to the following questions:

1. **How can we compute the probability of a given observation sequence O and a certain word w, $P(\mathbf{O}|\lambda_w)$?**
   This is required for comparing word hypotheses in classification.
2. **What is the most likely sequence that produced a given observation sequence ?**
   The knowledge about the most likely state sequence allows to assign timing information to the audio stream, It can be used for
   - ➜ the segmentation of speech data
   - ➜ a simplified training scheme for HMMs
   - ➜ decoding
3. **How can we choose the model parameters ?**
   Find the parameters for which maximum likelihood of the training data is achieved (**maximum likelihood estimation** = **MLE**). Other schemes for parameter estimation will be discussed next term.

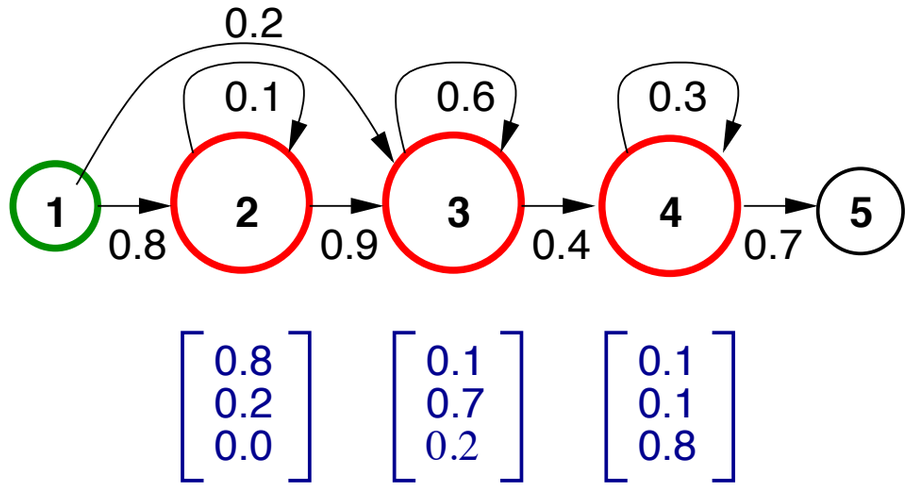
HMMs as generators

HMMs generate stationary random sequences. The procedure to generate a random sequence is:

1. Select an initial state according to the initial probabilities.
   If only one initial state with index i is possible the probability of starting in that state is one.
2. If the current state is emitting generate an output symbol according to the output distribution of the current state
3. Select the next state according to the transition probabilities from the current state. Move to the next state.
4. If the current state is emitting and does have other transitions than to itself repeat from 2.

Example:

State seq: $[1, 2, 3, 3, 4, 5]$
Observation seq: $[1, 1, 2, 3]$



First pick the state sequence and then map to observation sequence.

Computing the likelihood

If the HMM parameter λ_w, the state sequence $\mathbf{X}$ (path) and the observation sequence $\mathbf{O}$ are known the likelihood for the joint event can be expressed in terms of output probabilities and transitions:

$$P(\mathbf{O}, \mathbf{X}|\lambda_w) = p(\mathbf{O}|\mathbf{X}, \lambda_w)P(\mathbf{X}|\lambda_w)$$

The probability of the state sequence is similar as in Markov chains (the initial and final states, $x(0)$ and $x(T+1)$ are non-emitting)

$$P(\mathbf{X}|\lambda_w) = P(x(1)|x(0)) \prod_{t=1}^{T} P(x(t+1)|x(t))$$

The likelihood of the observation sequence $\mathbf{O}$ given the state sequence is

$$p(\mathbf{O}|\mathbf{X}, \lambda_w) = \prod_{t=1}^{T} p\left(\mathbf{o}_t|\mathbf{o}_{t-1}, \mathbf{o}_{t-2}, \ldots, \mathbf{o}_1, x(t)\right) = \prod_{t=1}^{T} p(\mathbf{o}_t|x(t))$$

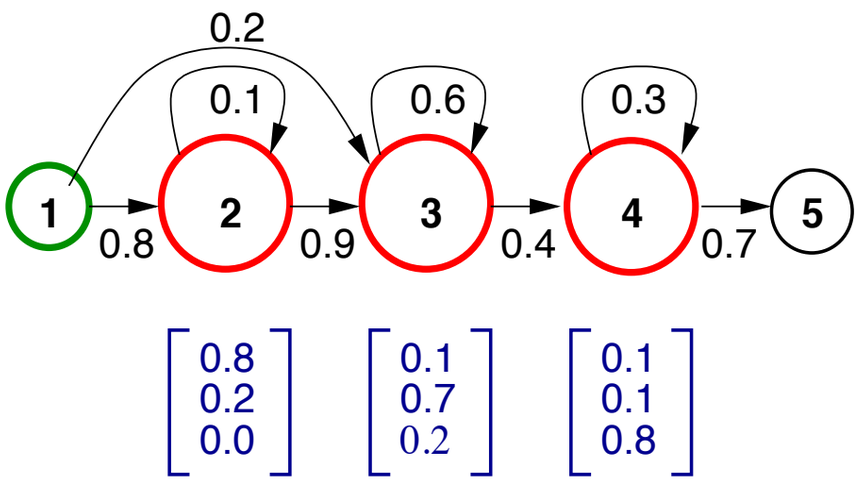
This makes use of the fact that an observation is independent from others, given the current state.

The "total" likelihood of the feature vector stream $\mathbf{O}$ given the model for a particular word λ_w can be computed by summation over all possible state sequences (**paths** !).

$$p(\mathbf{O}|\lambda_w) = \sum_{\mathbf{X}} p(\mathbf{O}, \mathbf{X}|\lambda_w) = \sum_{\mathbf{X}} p(\mathbf{O}|\mathbf{X}, \lambda_w) P(\mathbf{X}|\lambda_w)$$

where $\sum_{\mathbf{X}}$ denotes the summation over all possible sequences.

Example



Given the observation seq. $[1, 1, 2, 3]$ the possible state sequences (paths) are

$$\mathbf{X}_1 = [1, 2, 2, 3, 4, 5], \mathbf{X}_2 = [1, 2, 3, 3, 4, 5],$$
$$\mathbf{X}_3 = [1, 2, 3, 4, 4, 5], \mathbf{X}_4 = [1, 3, 3, 3, 4, 5],$$
$$\mathbf{X}_5 = [1, 3, 3, 4, 4, 5], \mathbf{X}_6 = [1, 3, 4, 4, 4, 5]$$

The total likelihood of this observation sequence is given by the sum over the joint likelihood of $\mathbf{O}$ and $\mathbf{X}_i$.

HMM assumptions

The definition of HMMs is based on **independence assumptions**. These are

➜ **Markov assumption**

The probability of a transition into state j from state i is independent of previous states. Using the transition probabilities $a_{ij} = P\left(x(t) = j | x(t-1) = i\right)$:

$$P(\mathbf{X}|\lambda_w) \quad = \quad a_{x(0)x(1)} \cdot \prod_{t=1}^{T} a_{x(t)x(t+1)}$$

➜ **Conditional independence**

Given a particular state at time t, the likelihood of the output symbol at that time is independent of any other state or previous output symbol

$$p(\mathbf{O}|\mathbf{X}, \lambda_w) \quad = \quad \prod_{t=1}^{T} p(\mathbf{o}_t|\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_{t-1}, \mathbf{X}, \lambda_w) = \prod_{t=1}^{T} b_{x(t)}(\mathbf{o}_t|\lambda_w)$$

The assumptions are problematic for modelling of speech since we expect smooth parameter trajectories $\Rightarrow$ strong correlation over time.

The most likely state sequence

If both the state sequence and the observation sequences are known a clear connection between any symbol in the vector stream and a state can be established. The feature vector stream also carries implicit information about time (e.g. one vector per 10ms). Ideally all vectors associated with one state should belong to the same quasi-stationary part of the speech signal. This allows to **segment** the signal in time.

However, the state sequence is **unknown** (hidden). One cannot derive the original sequence with certainty, but given an HMM λ_w and an observation sequence $\mathbf{O}$ the state sequence $\mathbf{X}^*$ with the highest likelihood can be computed:

$$\mathbf{X}^* = \arg\max_{\mathbf{X}} p(\mathbf{O}, \mathbf{X}|\lambda_w)$$

where the likelihood is given by

$$p(\mathbf{O}, \mathbf{X}^*|\lambda_w) = a_{x^*(0)x^*(1)} \cdot \prod_{t=1}^{T} a_{x^*(t)x^*(t+1)} b_{x^*(t)}(\mathbf{o}_t)$$

Note that $p(\mathbf{O}, \mathbf{X}^*|\lambda_w)$ can serve as an approximation to the total likelihood:

$$p(\mathbf{O}|\lambda_w) = \sum_{\mathbf{X}} p(\mathbf{O}, \mathbf{X}|\lambda_w) \approx p(\mathbf{O}, \mathbf{X}^*|\lambda_w)$$

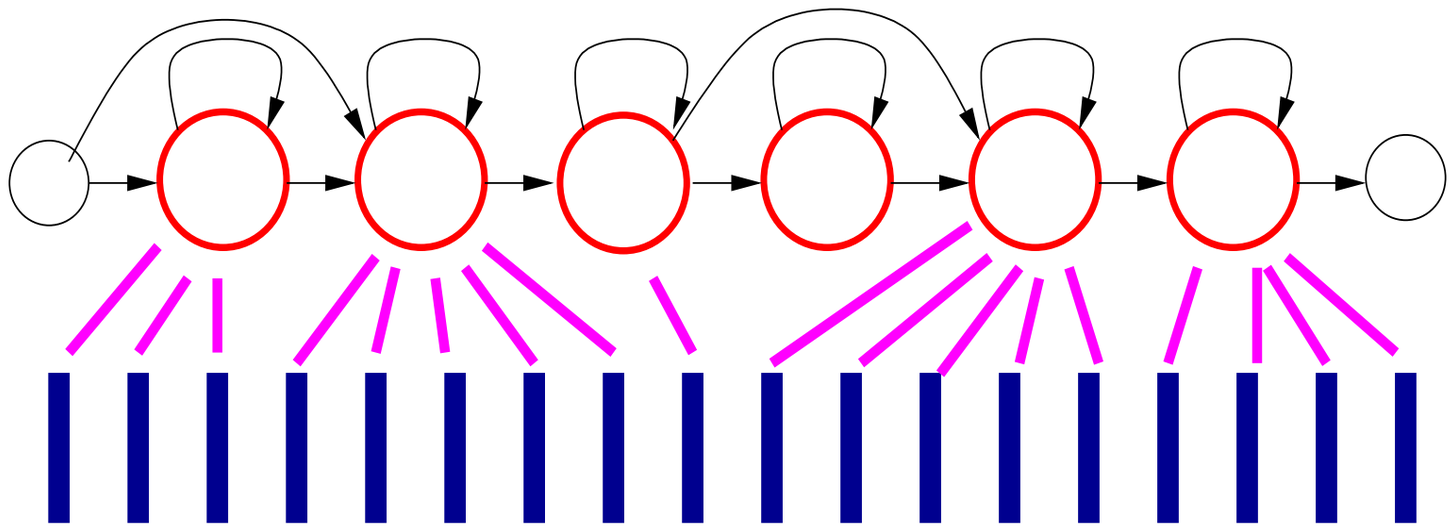
In that case all other paths (competing state sequences) are assumed to provide only a minor contribution to the total likelihood.

The knowledge about the best state sequence allows to temporally segment speech data. This can be used to

➜ segment words into smaller units (sub-word units).

➜ segment sentences into words

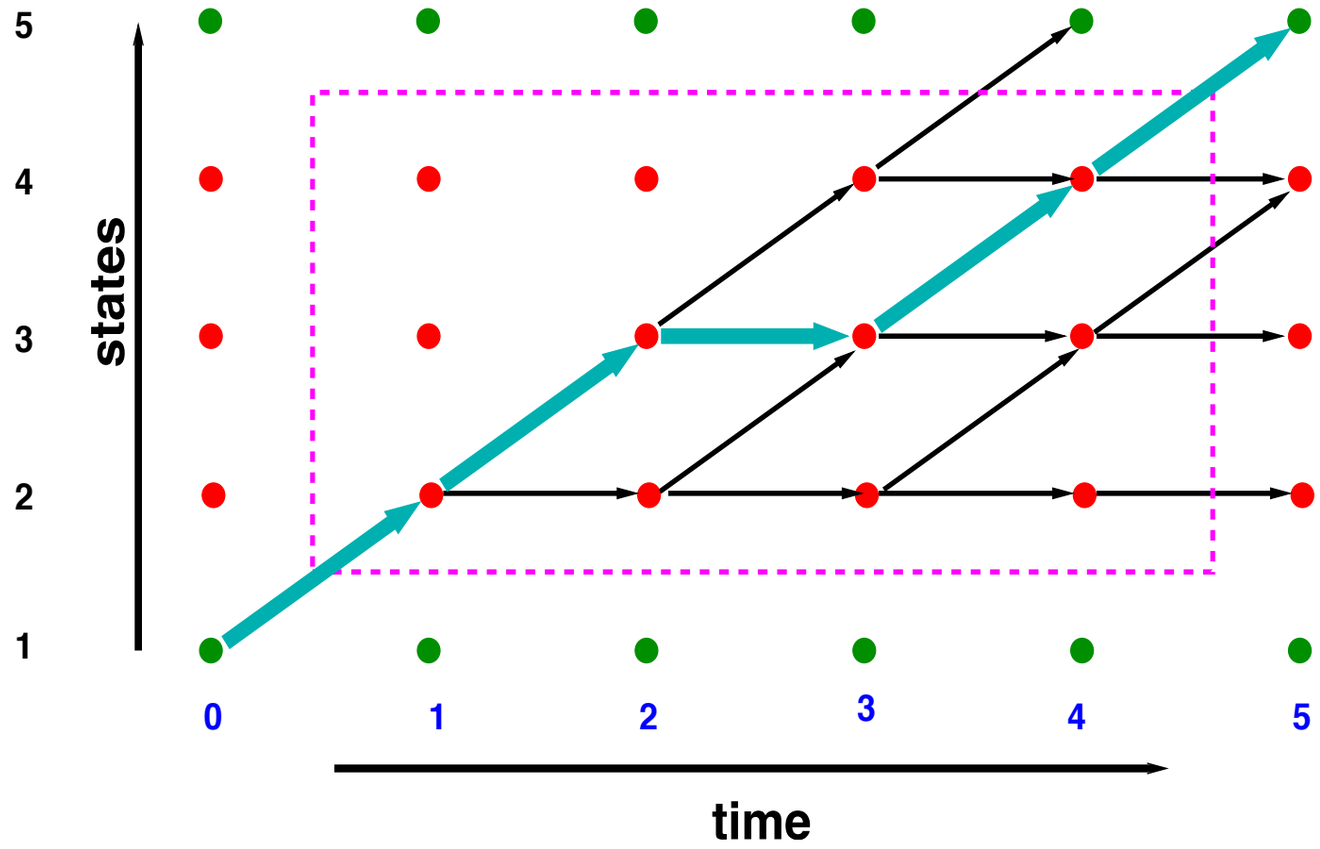
➜ to enable a simplified training procedure of HMM parameters



The Trellis

Another option to visualise the operations of an HMM is the trellis. The advantage of this type of characterisation is a joint visualisation with the observation vectors over time. This allows to plot the path through the HMM.

Example
A trellis for a 5 state HMM (initial and final states non-emitting) and an observation seq. of length 4.



The likelihood of a path through the trellis (e.g. the path depicted with bold arrows) can be computed by multiplication of the transition probabilities placed at the arcs and the output probabilities which are placed at the nodes.

Viterbi algorithm

The Viterbi algorithm is a **dynamic programming** procedure to obtain the most likely path for given model λ and observation sequence $\mathbf{O}$ of length T.

Define the likelihood of a **partial path** of length t which ends in state j:

$$\phi_j(t) = p(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t, x(1), \ldots, x(t-1), j|\lambda)$$

Then the likelihood of the partial path can be expressed in terms of the likelihood at the preceding state $x(t-1)$:

$$\phi_j(t) = a_{x(t-1),j} \cdot b_j(\mathbf{o}_t) \cdot \phi_{x(t-1)}(t-1) \tag{3}$$

Assume that our algorithm starts in the final state as this state necessarily belongs to the optimal path $\mathbf{X}^*$. Use eq. (3) to express the joint likelihood $\phi_N(T+1) = P(\mathbf{O}, \mathbf{X}^*|\lambda)$ in terms of the previous state:

$$\phi_N(T+1) = a_{x(T),N} \cdot \phi_{x(T)}(T) \tag{4}$$

Multiple paths may lead to the final state. However, the optimal path is defined as the one which achieves the highest likelihood value $p(\mathbf{O}, \mathbf{X}^*|\lambda)$.

Thus one can select the preceding state $x^*(T)$ such that $\phi_N(T+1)$ is maximal.

$$x^*(T) = \arg \max_{1 \leq j < N} [a_{jN} \cdot \phi_j(T)]$$

For simplicity we use the notation $\kappa = x^*(T)$. Once the state at time T is known the the partial likelihood $\phi_\kappa(T)$ can be expressed in terms of the predecessor state (as in eq(4)) using eq (3):

$$\phi_\kappa(T) = a_{x(T-1),\kappa} \cdot b_\kappa(\mathbf{o}_T) \cdot \phi_{x(T-1)}(T-1)$$

Note that this includes the value for the output distribution at time T. Again pick the preceding state that maximises $\phi_\kappa(t)$:

$$x^*(T-1) = \arg \max_{1 \leq j < N} [a_{j\kappa} \cdot b_\kappa(\mathbf{o}_T) \cdot \phi_j(T-1)]$$

The procedure can be repeated for $x^*(T-2), x^*(T-3), \ldots, x^*(1), x^*(0)$.

Note that in order to operate in this manner the values for partial paths, $\phi_j(t)$, need to be precomputed $\Rightarrow$ leads to a forward pass followed by "trace-back".

The Viterbi algorithm

Assume that HMM M has N states, but the states 1 and N are the non-emitting entry and exit states. The steps to obtain the most likely path $\mathbf{X}^*$ and the associated likelihood are

Initialisation

$\phi_1(0) = 1.0$

$\phi_j(0) = 0.0 \quad \text{for} \quad 1 < j < N$ and $\phi_1(t) = 0.0 \quad \text{for} \quad 1 \leq t \leq T$

Recursion

`for` $t = 1, 2, \ldots, T$

. . . `for` $j = 2, 3, \ldots, N-1$

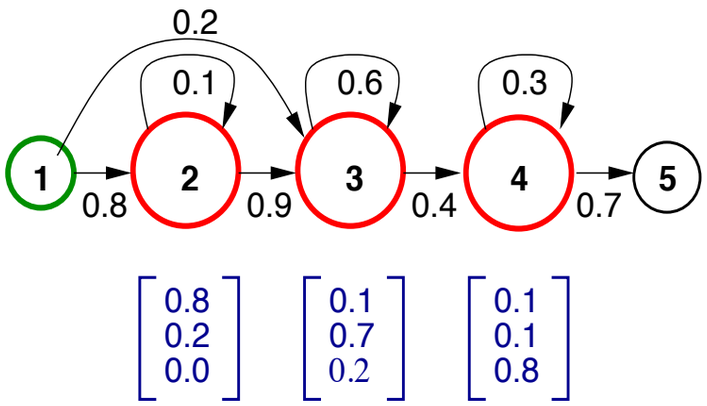
. . . . . compute $\phi_j(t) = \max_{1 \leq k < N} \left[\phi_k(t-1) a_{kj}\right] b_j(\mathbf{o}_t)$

. . . . . store the predecessor node: $\text{pred}_k(t)$

Termination $$p(\mathbf{O}, \mathbf{X}^* | \lambda) = \max_{1<k<N} \phi_k(T) a_{kN}$$

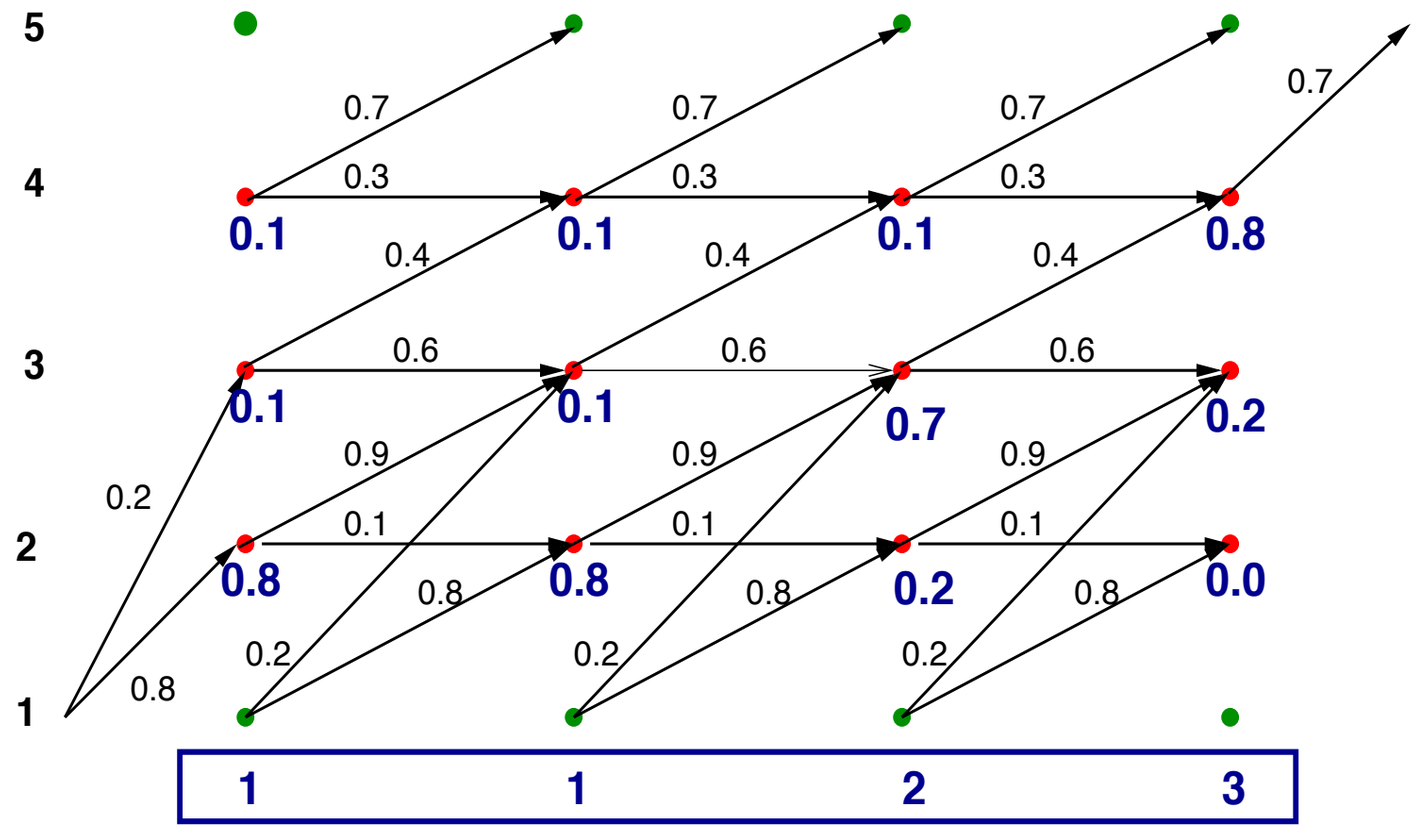
The most likely path can be recovered by **tracing back** using the predecessor information stored at each node $\text{pred}_k(t)$.

Viterbi - Example



$$\begin{bmatrix} 0.8 \\ 0.2 \\ 0.0 \end{bmatrix} \quad \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix} \quad \begin{bmatrix} 0.1 \\ 0.1 \\ 0.8 \end{bmatrix}$$

Given the HMM with discrete output distributions and the observed sequence $\mathbf{O} = [1, 1, 2, 3]$:



5	-	-	-	-	-	0.0072253
4	0.0	0.0	0.00080	0.002304	0.0103220	-
3	0.0	0.02	0.05760	0.032256	0.0038707	-
2	0.0	0.64	0.0512	0.001024	0.0	-
1	1.0	0.0	0.0	0.0	0.0	-
	-	1	1	2	3	-

Viterbi algorithm - Efficiency

Important properties of the algorithm are:

- ➜ The algorithm is efficient (local decisions)
- ➜ Paths merge and divide at roughly the same rate.
- ➜ Time for the search is linear in the length of the observed data, T

The direct calculation of the likelihood will cause arithmetic underflow. Thus in practice, an implementation of the algorithm is based on computation of $\log\left[p(\mathbf{O}, \mathbf{X}^*|\lambda)\right]$

$$\phi_j(t) = \max_{1 \leq k < N} \left[\log\left(\phi_k(t-1)\right) + \log\left(a_{kj}\right)\right] + \log\left(b_j(\mathbf{o}_t)\right)$$

In this formulation the search for the best state sequence appears as generalisation of the template matching procedures used in Dynamic Time Warping (DTW). DTW templates only represent a strict left-to-right topology. The number of frames in the template is equal to the number of states and all non-zero transition probabilities are equal.

HMM training

The objective in training is the estimation of all HMM parameters. The set parameters of an HMM λ using Gaussian output distributions is given by

$$\lambda = \{\{a_{ij}\}, \{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}\}$$

The parameters can be estimated from the training data. The training data is a set of R utterances $\mathbf{O}^{(r)}$ for the word w represented by λ. The most important criterion used for estimation of the HMM parameters tries to **maximise the likelihood** of producing the training data (maximum likelihood = ML). The likelihood of the set of utterances is given by

$$\prod_{r=1}^{R} p(\mathbf{O}^{(r)}|\lambda)$$

A solution which guarantees the global maximum is not known (apart from degenerate cases). However two iterative procedures (re-estimation steps) for ML estimation of HMM parameters exist:

➜ **Viterbi training** ➜ **Baum-Welch re-estimation**

In the case of Baum-Welch re-estimation an increase in likelihood is guaranteed (local optimality).

Viterbi Training

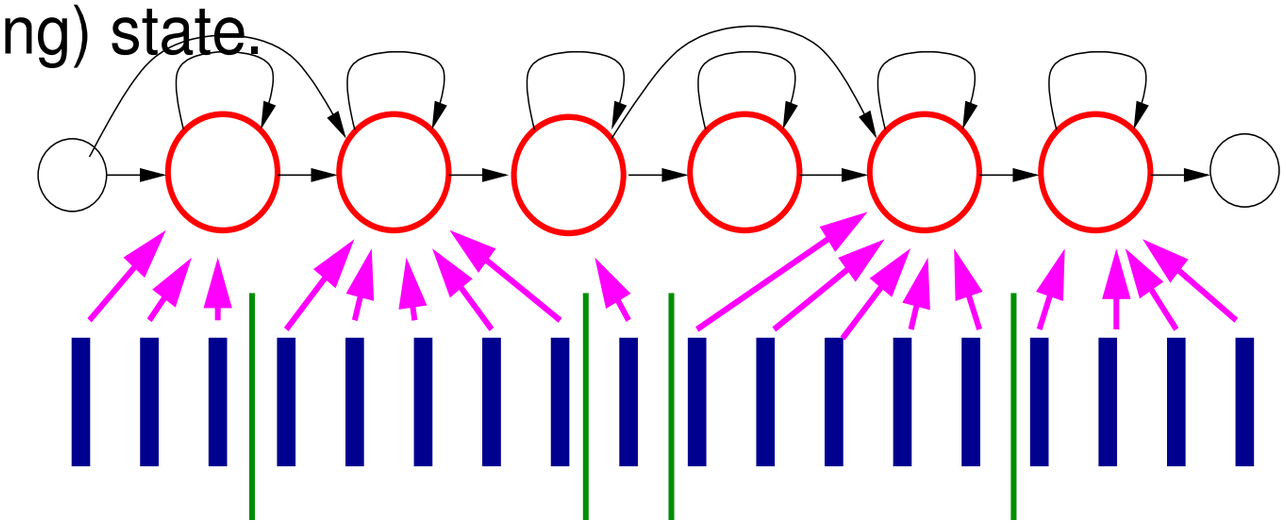
In the case of a Markov chain the knowledge about the state sequence allows to compute ML estimates for the transition probabilities:

$$a_{ij} = \frac{\text{Number of transitions state } i \rightarrow \text{ state } j}{\text{Number of transitions from state } i}$$

However for HMMs the state sequence is unknown (hidden). As was shown before the knowledge about the most likely state sequence $\mathbf{X}^*$ allows to approximate the total likelihood, i.e.

$$p(\mathbf{O}|\lambda) \approx p(\mathbf{O}, \mathbf{X}^*|\lambda)$$

The knowledge of the state sequence $\mathbf{X}^*$ clearly assigns each observation vector to exactly one (emitting) state.



Viterbi Training - Parameter estimation

Due to the conditional independence assumption the parameters of each output distribution can be estimated independently using the data vectors assigned to a state. Formally we can use an indicator variable $\delta(\cdot)$ to show if the observation vector at time t, $\mathbf{o}_t$, is associated with state j:

$$\delta(x^*(t) = j) = \begin{cases} 1 & \text{if } x^*(t) = j \\ 0 & \text{else} \end{cases}$$

This allows to form an expression for mean vectors and covariance matrices of the Gaussian output distribution for all states:

for each emitting state $(1 < j < N)$:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T^{(r)}}\delta(x^{(r)^*}(t) = j)\mathbf{o}_t^{(r)}}{\sum_{r=1}^{R}\sum_{t=1}^{T^{(r)}}\delta(x^{(r)^*}(t) = j)}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T^{(r)}}\delta(x^{(r)^*}(t) = j)(\mathbf{o}_t^{(r)} - \hat{\boldsymbol{\mu}}_j)(\mathbf{o}_t^{(r)} - \hat{\boldsymbol{\mu}}_j)'}{\sum_{r=1}^{R}\sum_{t=1}^{T^{r}}\delta(x^{(r)^*}(t) = j)}$$

Extending Gaussian models

Gaussian models are not very powerful because they

1. have only a very small number of parameterrs to represent the underlying data.
2. can only deal with one-dimensional data.

However logical extensions exist:

➜ Multivariate Gaussian distributions:
The PDF value of a a vector $\mathbf{o}$, $p(\mathbf{o})$can be computed.

➜ Gaussian mixture models:
So called “mutli-modal” distributions can be represented.

Mixture Models

Mixture models can be seen as analogy to the k-means model as derived for the deterministic cases.

$$p(x) = \sum_{i=1}^{M} w_i p_i(x)$$

Naturally the PDF has to fulfil all requirements (integral, greater or equal 0). If it is assumed that the components of the mixture are PDFs themselves these conditions imply

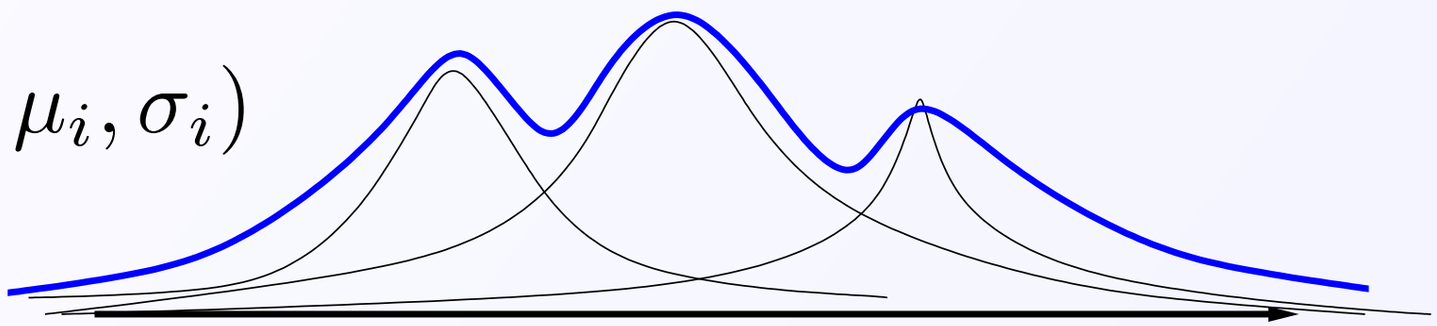
$$\sum_{i=1}^{M} w_i = 1$$

Gaussian Mixture Models

For a GMM the component distributions are Gaussian distributions. Each of these has two parameters, mean μ and variance σ^2. Thus an M component mixture model has $M \times *3$ parameters that need to learned from data.

In contrast to standard Gaussian models can capture multi-modal data (see below), similar to a k-means representation of the data. The model is easily extensible to higher dimensional spaces by employing multi-variate Gaussian distributions.

$$p(x) = \sum_{i=1}^{M} w_i \mathcal{N}(x, \mu_i, \sigma_i)$$



GMMs - Training

Training of GMMs using the maximum likelihood can in principle be understood

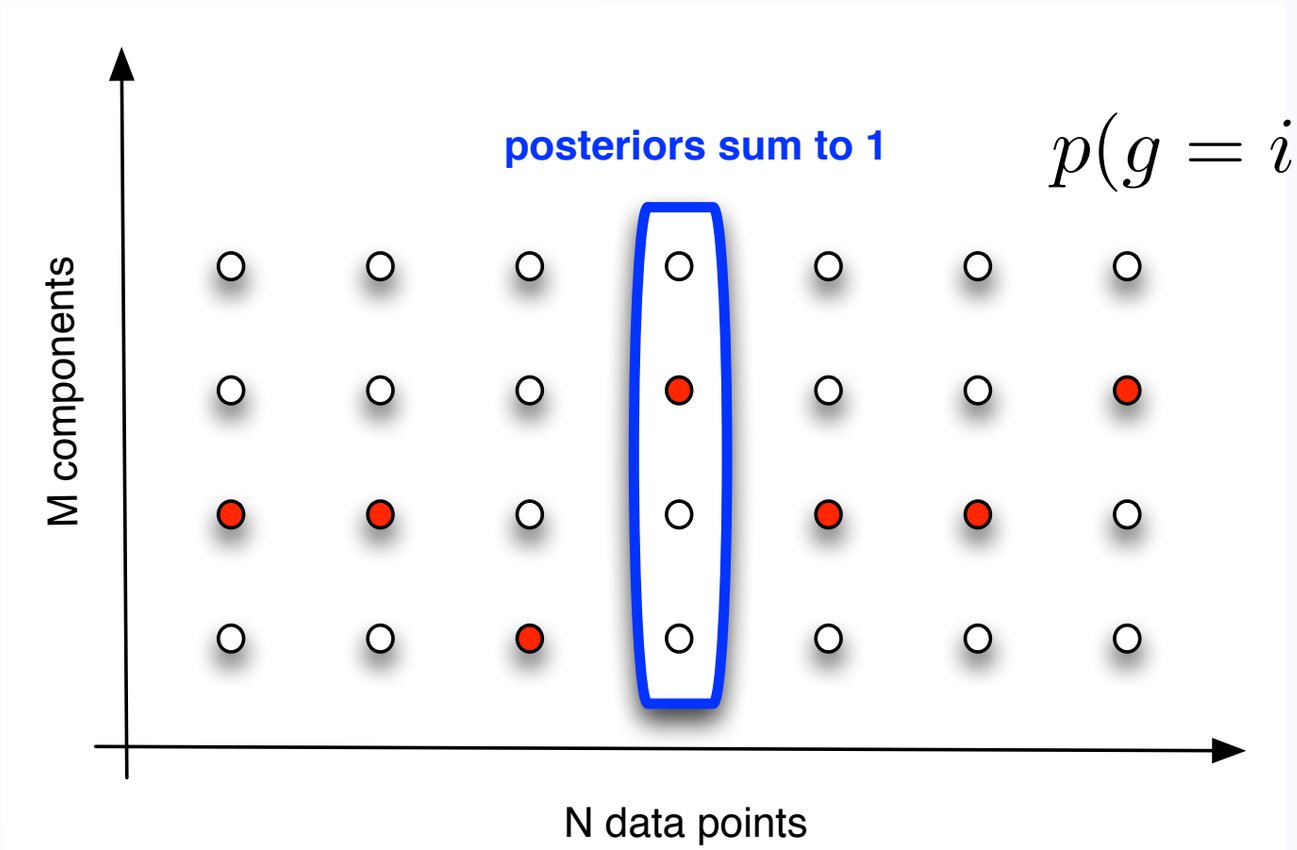
Recall: For k-menas clustering each training data sample was assigned to a centroid based on minimal distance. The centroids are recomputed based on the assigned data

Imagine that each data point was generated by one of the M components in a mixture model, we are just uncertain by which one. Thus in training we would need to uncover that assignment.

- In each iteration
  - find the Gaussain data is most probably the generator of the data sample
  - Re-estimate the Gaussian parameters based on the

GMM training

Decision theory requires to select the Gaussian with the highest posterior



$$p(g=i|o) = \frac{p(o|g=i)p(g=i)}{p(o)}$$

$$= \frac{p(o|g=i)p(g=i)}{\sum_{j=1}^{M} p(o|g=j)p(o)}$$

This is so-called Viterbi training !

GMM in recognition - 'Maturity' detection

- As before: one model for each class
- Decision based on posteriors

$$P(child|o) \lessgtr P(adult|o)$$

$$\Downarrow$$

$$p(o|child) \lessgtr p(o|adult)$$

- The model has now many more parameters
  - We can increase the number of parameters without changing the model structure:
    Scaleability

Baum-Welch re-estimation

Viterbi training avoids the problem of hidden state sequences by use of the segmentation provided by the most likely state sequence. The indicator variable used in the Viterbi re-estimation formulae

$$\delta(x^*(t) = j)$$

represents a **hard decision** that the observation $\mathbf{o}_t$ is produced by state j.

Baum-Welch re-estimation takes the uncertainty about the state sequence into account. The indicator variable in the re-estimation formulae of the output distribution parameters is replaced by the **state posterior probability**

$$L_j(t) = P(x(t) = j|\mathbf{O}, \lambda) \tag{5}$$

Any observation vector may occur with a certain probability at any time. The forward/backward algorithm is needed to provide an estimate for $L_j(t)$. Similarly the estimation of transition probabilities requires an estimate for

$$P(x(t) = i,\, x(t+1) = j|\mathbf{O}, \lambda)$$

A formal justification for this approach is found in the Expectation-Maximisation (E-M) algorithm which guarantees an increase in likelihood.

Forward algorithm

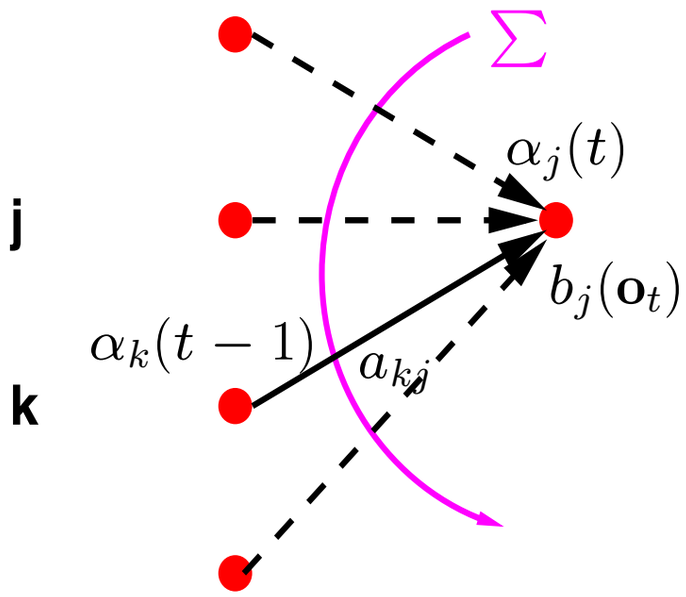
The forward algorithm allows to efficiently compute the total likelihood $p(\mathbf{O}|\lambda)$.

➜ Define the **forward variable** $\alpha_j(t) = p(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t, x(t) = j \,|\lambda)$

➜ A summation over all states at time t gives the total likelihood of the observation sequence up to time t: $P(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t|\lambda) = \sum_{j=1}^{N} \alpha_j(t)$

The likelihood of arriving in state j after t observations can be expressed in terms of the likelihood of being in state k at time $t-1$ (i.e. $\mathbf{o}_{t-1}$ is produced by state k):

$$p(\mathbf{o}_1, \ldots, \mathbf{o}_t,\, x(t-1) = k,\; x(t) = j \,|\lambda) = \alpha_k(t-1) a_{kj} b_j(\mathbf{o}_t)$$



A summation over all states at time $t-1$ allows allows to recursively compute $\alpha_j(t)$:

$$\alpha_j(t) = \sum_{k=1}^{N} p(\mathbf{o}_1, \ldots, \mathbf{o}_t,\, x(t-1) = k,\; x(t) = j \,|\lambda)$$

Forward algorithm - Steps

The forward algorithm allows to efficiently compute the total likelihood. Given an HMM with non-emitting entry and exit states 1 and N.

Initialisation

$$\alpha_1(0) = 1.0 \quad \text{and} \quad \alpha_j(0) = 0 \quad \text{for} \quad 1 < j \leq N$$

Recursion

`for` $t = 1, 2, \ldots, T$

`...for` $j = 2, 3, \ldots, N-1$

$$\alpha_j(t) = b_j(\mathbf{o}_t) \left[\sum_{k=1}^{N-1} \alpha_k(t-1) a_{kj} \right]$$

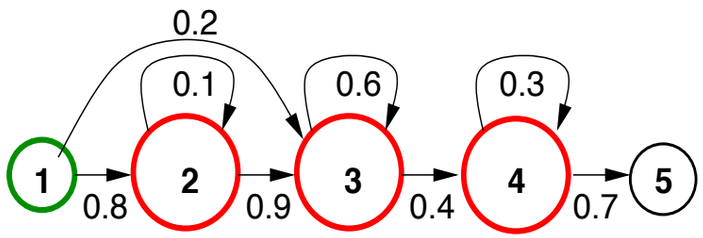
Termination

$$p(\mathbf{O}|\lambda) = \sum_{k=2}^{N-1} \alpha_k(T) a_{kN}$$

Note the similarity to the Viterbi algorithm. The forward algorithm can be used for recognition (at least for simple tasks):

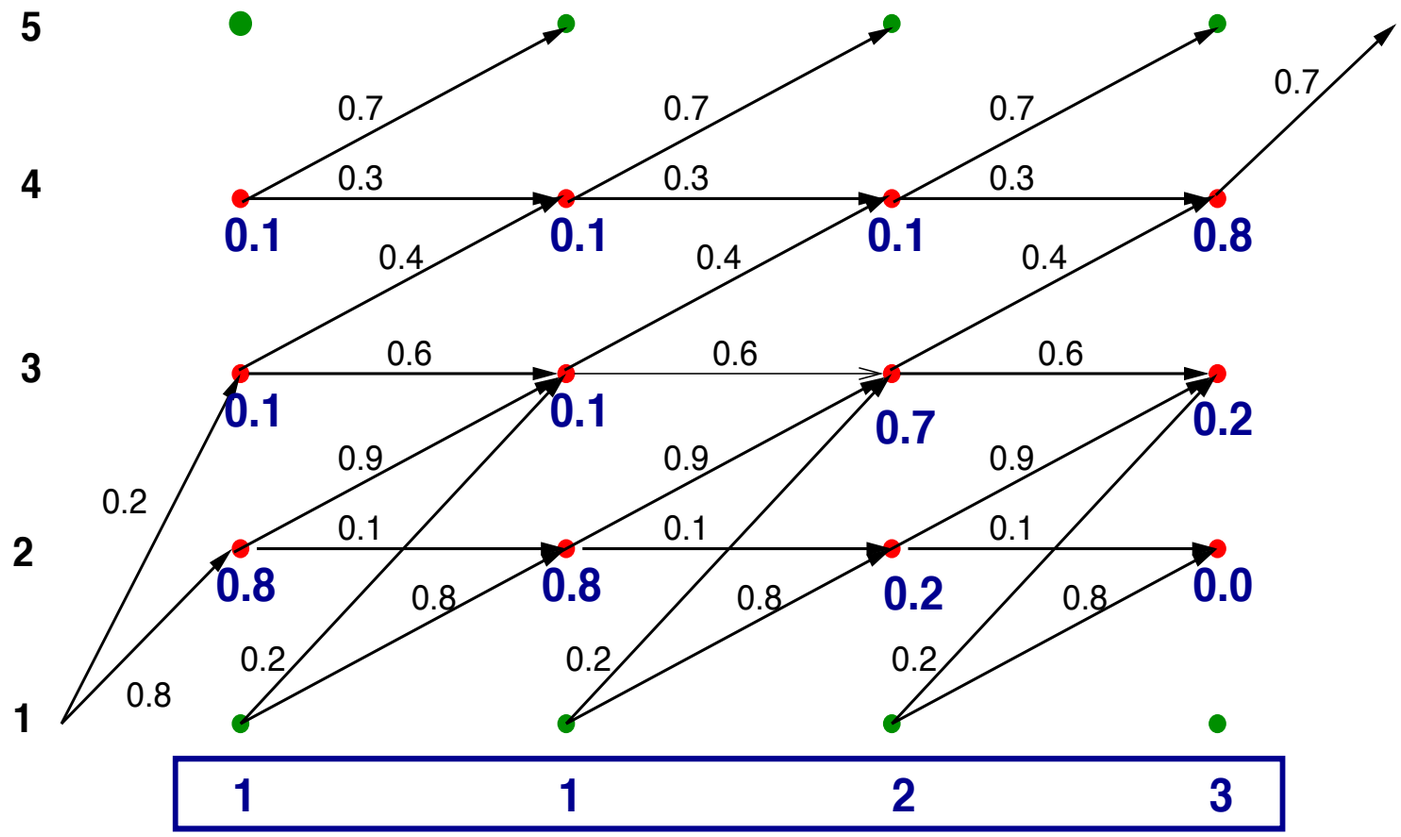
$$\hat{w} = \arg\max_w p(\mathbf{O}|\lambda_w) P(w)$$

Example (Forward algorithm)



$$\begin{bmatrix} 0.8 \\ 0.2 \\ 0.0 \end{bmatrix} \quad \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix} \quad \begin{bmatrix} 0.1 \\ 0.1 \\ 0.8 \end{bmatrix}$$

Given the HMM with discrete output distributions and the observed sequence $\mathbf{O} = [1, 1, 2, 3]$:



5	-	-	-	-	-	0.013156
4	0.0	0.0	0.0008	0.002376	0.018795	-
3	0.0	0.02	0.0588	0.056952	0.0070186	-
2	0.0	0.64	0.0512	0.001024	0.0	-
1	1.0	0.0	0.0	0.0	0.0	-
	-	1	1	2	3	-

Calculating $L_j(t)$

in order to compute $L_j(t)$ it is necessary to define the **backward variable**

$$\beta_j(t) = p(\mathbf{o}_{t+1}, \mathbf{o}_{t+2}, \ldots, \mathbf{o}_T \,|x(t) = j, \lambda)$$

The definitions of forward and backward variables are **not symmetric** (**!**). Given both variables for a certain state j at time t

$$p(x(t) = j, \mathbf{O} \,|\lambda) = \alpha_j(t)\beta_j(t)$$

and therefore

$$L_j(t) = P(x(t) = j \,|\mathbf{O}, \lambda) = \frac{1}{p(\mathbf{O}|\lambda)}\alpha_j(t)\beta_j(t)$$

for re-estimation of the transition probabilities we need

$$p(x(t) = k, x(t+1) = j, \mathbf{O} \,|\lambda) = \alpha_k(t)a_{kj}b_j(\mathbf{o}_{t+1})\beta_j(t+1)$$

and the desired probability

$$P(x(t) = i, x(t+1) = j|\mathbf{O}, \lambda) = \frac{\alpha_i(t)a_{ij}b_j(\mathbf{o}_{t+1})\beta_j(t+1)}{p(\mathbf{O}|\lambda)}$$

Backward algorithm

Similar to the forward algorithm the backward variable can be computed efficiently using a recursive algorithm:

Initialisation

$\beta_j(T) = a_{jN} \qquad 1 < j \leq N$

Recursion

`for` $t = T-1, T-2, \ldots 2, , 1$

`...for` $j = N-1, N-2, \ldots, 1$

$$\beta_j(t) = \sum_{k=2}^{N} a_{jk} b_k(\mathbf{o}_{t+1}) \beta_k(t+1)$$

Termination

$$P(\mathbf{O}|\lambda) = \beta_1(0) = \sum_{k=2}^{N-1} a_{1k} b_k(\mathbf{o}_{t+1}) \beta_k(t+1)$$

Baum-Welch re-estimation - Formulae

For an N state HMM (states 1 and N are non-emitting) and Gaussian output distributions the parameters can be re-estimated using the update formulae:

➜ **Transition probabilities**

$$a_{ij} = \frac{\sum_{r=1}^{R} \frac{1}{p(\mathbf{O}^{(r)}|\lambda)} \sum_{t=1}^{T^{(r)}} \alpha_i(t) a_{ij} b_j(\mathbf{o}_{t+1}^{(r)}) \beta_j(t+1)}{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_i^{(r)}(t)} \quad \text{for} \quad 1 \leq i < N, \; 1 \leq j \leq N$$

$$a_{jN} = \frac{\sum_{r=1}^{R} L_j^{(r)}(T^{(r)})}{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_j^{(r)}(t)} \quad \text{for} \quad 1 \leq j < N$$

➜ **Output distribution parameters (Gaussian)**

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_j^{(r)}(t) \mathbf{o}_t^{(r)}}{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_j^{(r)}(t)}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_j^{(r)}(t) (\mathbf{o}_t^{(r)} - \hat{\boldsymbol{\mu}}_j)(\mathbf{o}_t^{(r)} - \hat{\boldsymbol{\mu}}_j)'}{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_j^{(r)}(t)} \quad \text{for} \quad 1 < j < N$$

Note that multiple iterations of estimating the parameters are required.