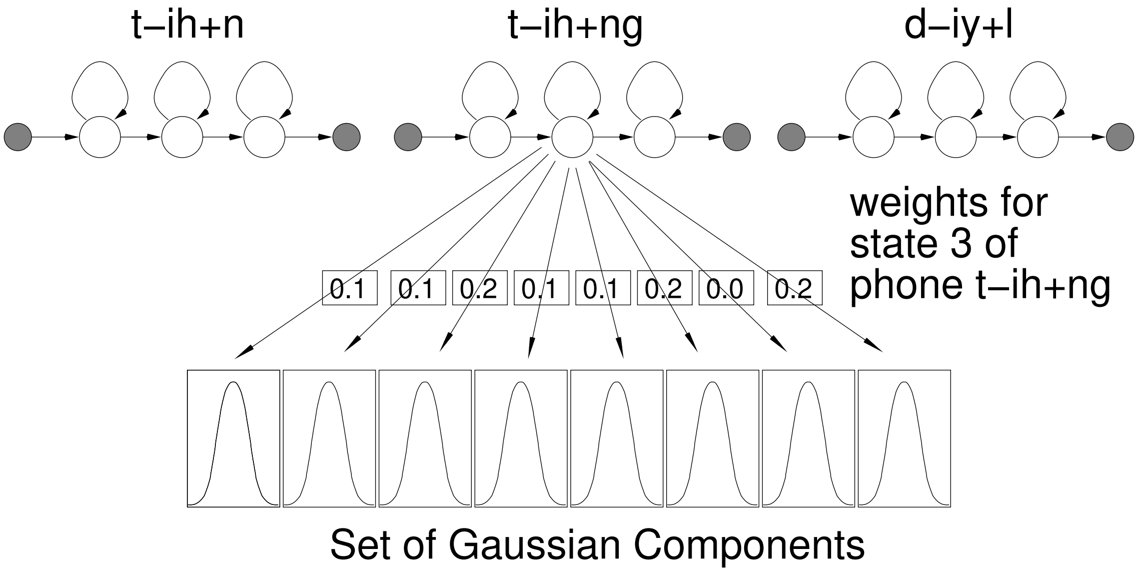
University of Cambridge

MPhil in Computer Speech Text & Internet Technology

Module: Speech Processing II

Lecture 3: Hidden Markov Models II



Lent 2003

Likelihood Calculation

Normally (so that underflow is avoided) *log-likelihoods* are calculated. Consider calculating the log-likelihood of a particular Gaussian component m of state j

$$\begin{aligned}\log(\mathcal{N}(\mathbf{o};\boldsymbol{\mu}_{jm},\boldsymbol{\Sigma}_{jm})) &= \mathcal{L}(\mathbf{o};\boldsymbol{\mu}_{jm},\boldsymbol{\Sigma}_{jm}) \\ &= -\frac{1}{2}\log\left(|\boldsymbol{\Sigma}_{jm}|(2\pi)^n\right) \\ &\quad -\frac{1}{2}(\mathbf{o}-\boldsymbol{\mu}_{jm})'\boldsymbol{\Sigma}_{jm}^{-1}(\mathbf{o}-\boldsymbol{\mu}_{jm})\end{aligned}$$

Only the second term is dependent on the observed data, so replace the first term by a constant determined during training, k_{jm}. In addition the *inverse* covariance matrix is stored (rather than the covariance matrix).

The total cost is then dependent on the form of the covariance matrix

Covariance	Number Mult. Acc.	Sub/Additions
Full	n^2+n	n
Diagonal	$2n$	n

Likelihood Calculation (cont)

However we required the state log-likelihood. Then we need

$$\begin{aligned}\log(b_j(\mathbf{o})) &= \log\left(\sum_{m=1}^{M} \mathcal{N}(\mathbf{o}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})\right) \\ &= \log\left(\sum_{m=1}^{M} c_{jm} \exp(\mathcal{L}(\mathbf{o}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}))\right)\end{aligned}$$

Addition in the log-domain if implemented directly is expensive (requires an exponential for each component and a log).

If log-likelihoods are calculated and stored, then the *logadd* is usually implemented using

$$\log(\exp(A) + \exp(B)) = A + \log(1 + \exp(B - A))$$

assuming that $A > B$. It is only worth computing the second term when $B - A$ exceeds some accuracy threshold.

So far the mixture weight has not been considered. This can be implemented with no overhead by combining the weight into the prestored constant as

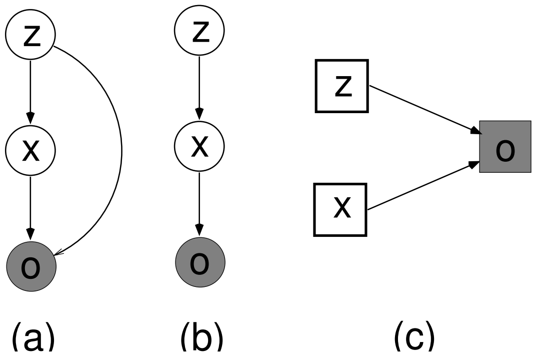
$$c_{jm} \exp(\mathcal{L}(\mathbf{o}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})) = \exp(\log(c_{jm}) + \mathcal{L}(\mathbf{o}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}))$$

To avoid the cost of the logadd some systems approximate the summation by a maximum

$$\log(b_j(\mathbf{o})) \approx \max_m \left\{\log(c_{jm}) + \mathcal{L}(\mathbf{o}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})\right\}$$

Bayesian Networks

When HMMs have been drawn, they have shown the state topology, much the same as a Markov chain, but little information about the indepence assumptions behind the model. One way of showing these indpendence assumptions is using a *Bayesian network*.

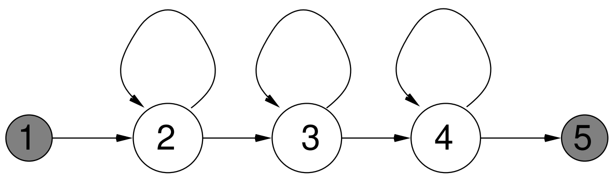


The above diagram shows three networks.

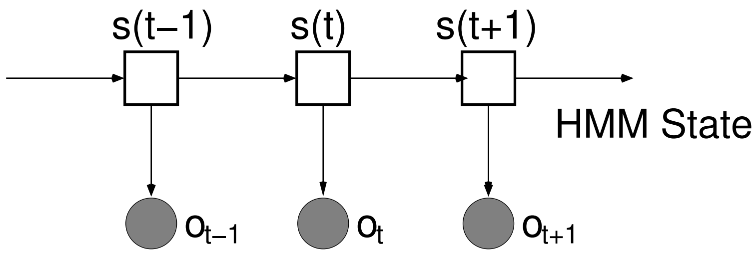
- **(a)** shows three continuous (circles) random variables. o is observed (shaded). The interpretation of this
$$p(z, x, o) = p(z)p(x|z)p(o|z, x)$$
- **(b)** shows three continuous (circles) random variables. o is observed (shaded). The interpretation of this
$$p(z, x, o) = p(z)p(x|z)p(o|x)$$
- **(c)** shows three discrete (squares) random variables. o is observed (shaded). The interpretation of this
$$P(z, x, o) = P(z)P(x)P(o|z, x)$$

Dynamic Bayesian Networks

For HMMs there is also a dynamic aspect of the model. This requires a *dynamic Bayesian network*. Rather than expressing an HMM as



It may be drawn as



Here the conditional independence assumptions behind the HMM, that observations are independent given the state, is clear. However the state topology is not shown.

Bayesian networks are very useful for graphically representing conditional independence. It is also possible to derive general re-estimation formulae for many topologies.

Discrete HMMs

HMMs can also be used to model sequences of discrete symbols. In this case, the output distribution is discrete.

$$b_j(q) = P(\hat{q}_t = q \,|s(t) = j)$$

where $\hat{q}_t$ is the observed symbol at time t.

The output distribution ($b_j(q)$):

- just a table of probabilities (effectively a histogram)
- estimated by simply counting the number of times each symbol occurs in each state, weighted by the probability of the HMM being in that state when the symbol occurs

  $$\hat{b}_j(q) = \frac{\Sigma_{t \in \{\hat{q}_t = q\}} L_j(t)}{\Sigma_{t=1}^{T} L_j(t)}$$

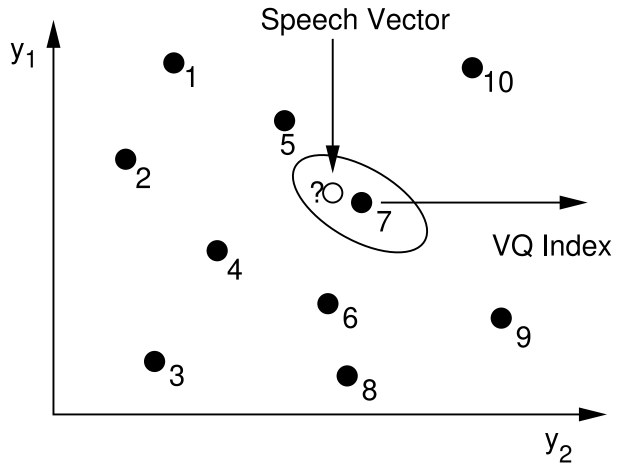
  A minimum value for the new estimate of $\hat{b}_j(q)$, eg 1e-5, must be set as if set to zero it is likely to cause recognition errors (or some other distribution smoothing).
- highly efficient during recognition - a simple table lookup.

Discrete HMMs can be used for modelling data which is naturally discrete. They can also be used in for continuous data by using a *vector quantiser*. Here, the continuous feature vector $\mathbf{o}_t$ is mapped to a discrete index $\hat{q}_t$.

Vector Quantisation

A vector quantiser consists of a *codebook* of Q (speech) vectors $\{\mathbf{v}_1 \ldots \mathbf{v}_Q\}$. The process is:

1. the incoming (speech) vector is compared to the set of codebook vectors;
2. vector is replaced by the index of the nearest codebook vector.



The vector quantisation process is illustrated above. A variety of distance measures may be used:

1. **Euclidean**: $d(\mathbf{o}, \mathbf{v}_q) = \sqrt{((\mathbf{o} - \mathbf{v}_q)'(\mathbf{o} - \mathbf{v}_q))}$
2. **Weighted Euclidean**: $d(\mathbf{o}, \mathbf{v}_q) = \sqrt{((\mathbf{o} - \mathbf{v}_q)'\mathbf{\Sigma}^{-1}(\mathbf{o} - \mathbf{v}_q))}$
3. **Mahalanobis**: $d(\mathbf{o}, \mathbf{v}_q) = (\mathbf{o} - \mathbf{v}_q)'\mathbf{\Sigma}^{-1}(\mathbf{o} - \mathbf{v}_q)$
4. **Minkowski metric, order** s: $d(\mathbf{o}, \mathbf{v}_q) = \sqrt[s]{(\Sigma_k |o_k - v_{qk}|^s)}$

Training the Quantiser

The **Generalised Lloyd** algorithm or Linde-Buzo-Gray (LBG) algorithm is commonly used to train the VQ code-book. It consists of :

Initialisation:

Choose an arbitrary set of Q vectors, $\{\mathbf{v}_1 \dots \mathbf{v}_Q\}$

Recursion:

1. For each speech frame $\mathbf{o}_t$ in the training data calculate index $\hat{q}_t$ where
$$\hat{q}_t = \arg\min_{Q} \{d(\mathbf{o}_t, \mathbf{v}_q)\}$$
2. Compute the total distortion that has occurred due to this quantisation
$$D = \sum_{t=1}^{T} d(\mathbf{o}_t, \mathbf{v}_{\hat{q}_t})$$
If D is sufficiently small, or sufficient iterations, **stop**
3. For each compute the centroid (and other parameters if required) of all indices
$$\mathbf{v}_q = \frac{\sum_{t \in \{\hat{q}_t = q\}} \mathbf{o}_t}{\sum_{t \in \{\hat{q}_t = q\}} 1}$$
Return to step 1.

Multiple Codebooks

Problems:

- quantisation error: error in approximating incoming vector by codebook entry;
- increasing number of codebook entries, increases resolution, but estimation problems;

Compromise is **multiple codebooks**:

- split feature vector into groups, e.g. static, delta and delta-delta parameters;
- generate a separate codebook for each group.

Consider 3 codebooks having 256 entries each:

- **Number of parameters**: $256 \times 3 = 768$
- **Number of possible indices**: $256^3 = 16777216$

Nothing is ever free - assumption that feature vector groups are independent of one another.

Discrete HMMs typically perform worse than continuous density HMMs.

Semi-Continuous HMMs

Semi-Continuous HMMs (SCHMMs), also called Tied-Mixture HMMs, are a kind of "half-way house" between CDHMMs and DHMMs. Here

$$b_j(\mathbf{o}) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(\mathbf{o}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$$

In contrast to CDHMMs, $\mathcal{N}(\mathbf{o}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ is *independent* of the state. Thus all states make use of the *same* set of Gaussian distributions, only the component weights are state specific.
Of course, the number of Gaussians is quite large as they need to be representative of all acoustic space.
The parameters of the system are:

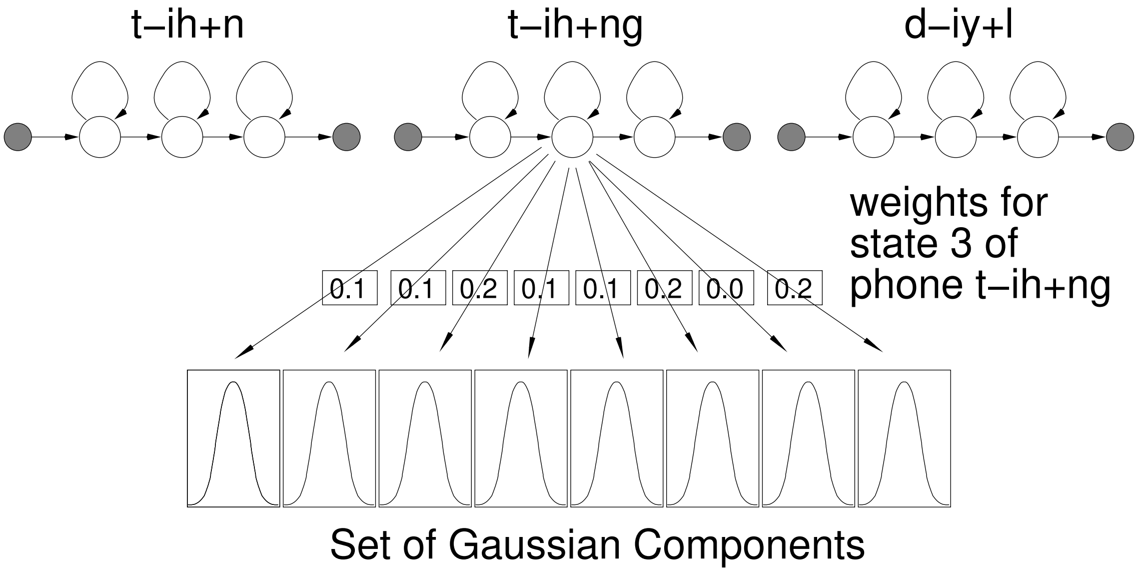
1. the transition matrix (standard HMM);

2. the set of means and variances for the Gaussian distributions

$$\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M\}$$

3. the set of component weights for all N_T states

$$\{\{c_{11}, \ldots, c_{1M}\}, \ldots, \{c_{N_T1}, \ldots, c_{N_TM}\}\}$$



The training of SCHMMs is very similar to training a standard single Gaussian HMM. The alignments are obtained in the same way as for standard CDHMMs, thus

$$L_{jm}(t) \; = \; P(s(t) = jm|\mathbf{O}, \mathcal{M})$$

In addition it is necessary to obtain the posterior probability of emitting a particular vector from a particular component (independent of the state), this is

$$L_m(t) = \sum_{j=1}^{N} L_{jm}(t)$$

The component weights are obtained identically to the multiple component CDHMM case

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} L_{jm}(t)}{\sum_{t=1}^{T} L_j(t)}$$

and the means and variances are based on the component alignments

$$\hat{\boldsymbol{\mu}}_m = \frac{\sum_{t=1}^{T} L_m(t)\mathbf{o}_t}{\sum_{t=1}^{T} L_m(t)}$$

SCHMMs can be more efficient than CDHMMs at run-time. The probabilities may be calculated as

1. Calculate $b_m(\mathbf{o}_t)$ for all components. Sort to obtain the top k components, all others are set to some small value.
2. For each state calculate
$$b_j(\mathbf{o}_t) = \sum c_{jm} b_m(\mathbf{o}_t)$$
where the sum is only over the top (ranked by log-likelihood) k components.

This can give large savings as the probability for each Gaussian is only calculated once, and typical values are $M = 256$, $k = 4$.

Multiple streams are often used with SCHMMs. The reasons for these are the same as those described in the previous section for using multiple codebooks with DHMMs. Thus the number of streams is equivalent to the number of codebooks and the number of components in each stream is equivalent to the number of codewords.

Note the analogy between the VQ symbol k and the Gaussian m, and b_{jk} and the component weights c_{jm} in an SCHMM.

Multiple Training Examples

The training equations given so far have only been for a single training utterance. In general, multiple training examples are used. This has a slight effect on the re-estimation formulae (consider only CDHMMs) as summations on numerators and denominators of re-estimation formulae are taken over all training sequences.

Data:

$$\left\{\mathbf{O}^1, \ldots, \mathbf{O}^R\right\}, \quad \mathbf{O}^r = \{\mathbf{o}_1^r, \ldots, \mathbf{o}_{T^r}^r\}$$

1. **Transition Probabilities**

$$\hat{a}_{ij} = \frac{\sum_{r=1}^{R} \frac{1}{p(\mathbf{O}^r|\mathcal{M})} \sum_{t=1}^{T^r-1} \alpha_i^r(t) a_{ij} b_j(\mathbf{o}_{t+1}^r) \beta_j^r(t+1)}{\sum_{r=1}^{R} \frac{1}{p(\mathbf{O}^r|\mathcal{M})} \sum_{t=1}^{T^r-1} \alpha_i^r(t) \beta_i^r(t)}$$

2. **Output Probability Distributions**

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

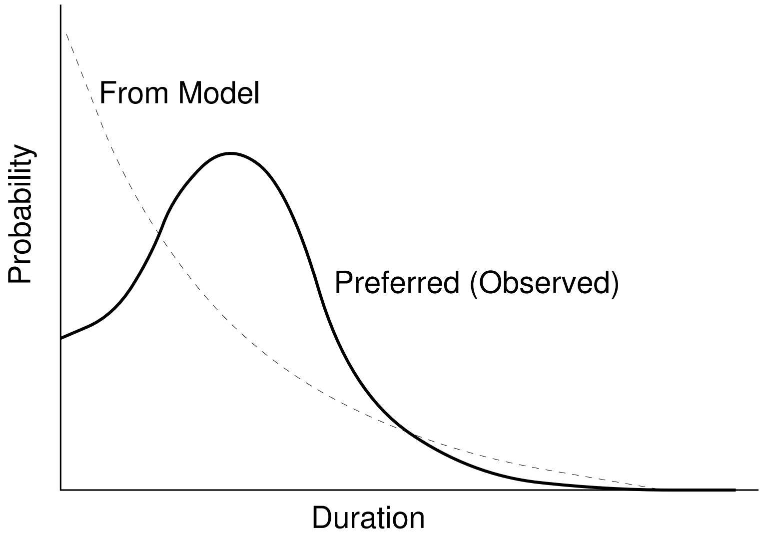
$$\hat{\sigma}_{jk}^2 = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T^r} L_j^r(t) (o_{tk}^r - \hat{\mu}_{jk})^2}{\sum_{r=1}^{R} \sum_{t=1}^{T^r} L_j^r(t)}$$

Duration Modelling

The problem of **duration modelling**, or how long should be spent in a state, has not been addressed. From the state transition probabilities

$$d_i(\tau) = a_{ii}^{\tau-1}(1 - a_{ii})$$

where $d_i(\tau)$ is the probability of staying in state i for τ frames.



Using the durations obtained directly from the transition probabilities is a poor model of the actual duration.
Duration modelling is important in order to make some distinctions e.g.

league vs *leek*

where the principle difference is in the length of the vowel.

There are problems associated with duration modelling. The duration in each state is dependent on the duration spent in the surrounding states. It is a function of the speaker rate.

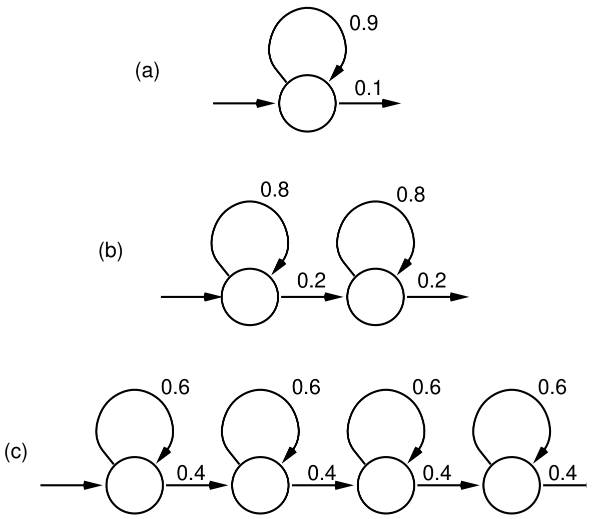
The duration may be considered as a separate knowledge source from the acoustic vector probability, and as such may require some form of information weighting to balance the acoustic and duration information.

Three methods for improving duration modelling.

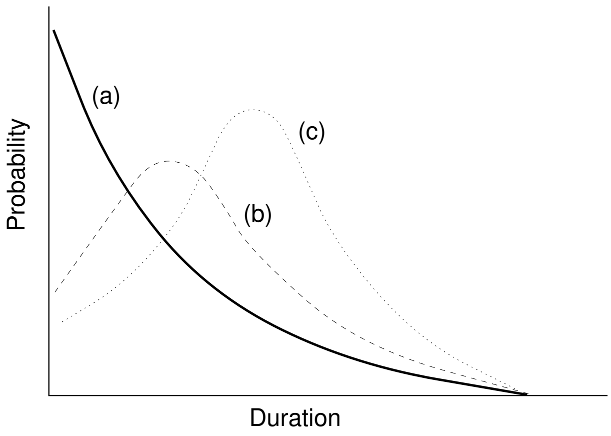
1. Use a number of states where the output probability distribution is tied.
2. Integrate a parametric model of the state duration
3. Duration models applied as a *post-processing* stage.

Using Additional States

Additional states are added where the same output distribution is tied over all states.



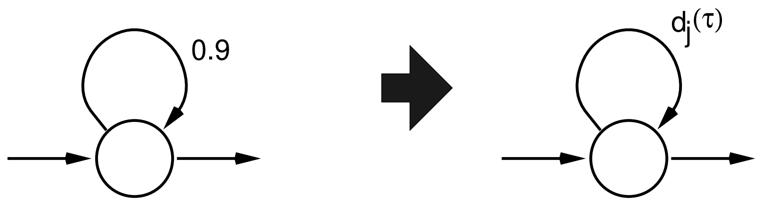
If $d(\tau)$ is calculated over **all** paths through the model then the following set of distributions are obtained.



If just the most likely state path is obtained through the model then all distributions are similar to (a).

Parametric Duration Models

Replace the self loop transition probability by a distribution $d_j(\tau)$, thus



Unfortunately this is no longer a Markov process, it is a **Semi-Markov process**. The probability calculations are complicated as it is now dependent on the time spent within the state. The forward probability becomes

$$\alpha_i(t) = \sum_{\tau \leq t} \sum_{j=1}^{N} \alpha_j(t-\tau) a_{ji} d_j(\tau) \prod_{\theta=1}^{\tau} b_j(\mathbf{o}_{t-\tau+\theta}))$$

Similar forms for the backward probability and re-estimation formulae may be derived. Both *Poisson* and *Gamma* distributions have been used for $d_i(\tau)$.

It is simple to see the additional computational overhead in the above probability calculation. As all previous possible durations must be examined the computational load for an observation of length T has increased from $\mathcal{O}(T)$ to $\mathcal{O}(T^2)$. This may be reduced by setting a *maximum duration*.

Post-processing Duration Modelling

For isolated word recognition a simple post-processor method can be effective in modelling state duration (or model/word duration).

1. Calculate $\log(p(\mathbf{O}|\mathcal{M}_i))$ for each model
2. *backtrack* state-by-state to find normalised durations, τ'_j
$$\tau'_j = \frac{\tau_j}{T}$$
where τ_j is the actual number of frames in state j and T is the total number of frames in the utterance.
3. Make recognition decision based on
$$\log(\hat{p}(\mathbf{O}|\mathcal{M}_i)) = \log(p(\mathbf{O}|\mathcal{M}_i) + \delta \sum_{j=1}^{N} \log(p_j(\tau'_j))$$

where δ is a weighting constant and $p_j(\tau'_j)$ is the probability density function for the normalised duration of state j. The duration may be modelled parametrically (Gamma distribution) or as a histogram.

This method has the advantage that normalised durations, as opposed to absolute durations are used. This, crudely, makes some account of the global speaker rate. Additionally it is highly efficient, compared with the direct on-line duration modelling.

Unfortunately there are disadvantages.

1. The initial estimates of the state durations are made with no reference to a duration model. Incorporating a duration model may alter the state durations, hence normalised durations and the final probabilities.

2. There is no simple extension of this to continuous speech recognition, unless alternative recognition strings are available. These may be obtained using an *N-Best* algorithm or via the use of *word lattices*. In these cases some improvements have been reported in some recognition systems.