MPhil in Machine Learning and Machine Intelligence

Module MLMI2: Speech Recognition

L5: HMM Speech Recognition—Baum-Welch Training and GMMs

Phil Woodland
pcw@eng.cam.ac.uk

Michaelmas 2021

Cambridge University Engineering Department

Outline

These lecture addresses the standard **maximum likelihood techniques** for the estimation of the model parameters of an HMM set as an alternative to **Viterbi training** that we have already seen.
This lecture will cover parameter estimation:

- Forward-Backward Algorithm
- Baum-Welch Re-estimation

to get estimates of the HMM parameters:

- transition probabilities
- mean vectors
- covariance matrices

We will start assuming that we will train an HMM for an isolated unit on a portion of speech data and finally discuss issues with training for continuous speech recognition.

We will then discuss extending the output distribution types to using **Gaussian mixture models** (GMM-HMMs) as state output distributions and discuss parameter training in this case also.

Baum-Welch re-estimation

Viterbi training avoids the problem of summing over hidden state sequences by using 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 i.e. the probability of the HMM being in a particular state at each time.

Hence, the indicator variable in the re-estimation formulae is replaced by the **state posterior probability**

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

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

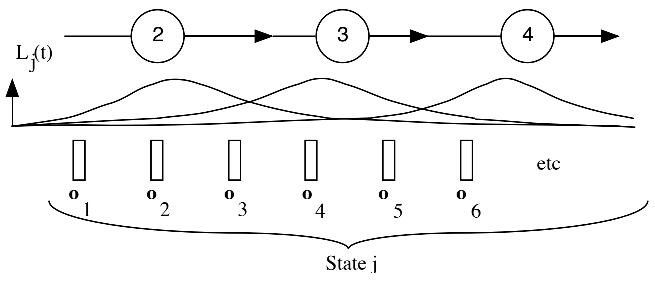
$$P(x(t) = i,\ x(t+1) = j|\mathbf{O}, \lambda)$$

A formal justification for this approach is found in the **Expectation-Maximisation (E-M)** algorithm which guarantees an increase in likelihood on each iteration (the posterior probabilities depend on the parameters) unless at a local likelihood maximum.

Baum-Welch Re-Estimation Formulae

Assume:

- HMM could be in **any** state at time t
- probability of being in state j at time t is $L_j(t)$



For single training example case, parameters can be estimated as weighted averages.

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

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} L_j(t)[(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_j)']}{\sum_{t=1}^{T} L_j(t)}$$

These equations are called the **Baum-Welch Re-Estimation Formulae** and are an example of the E-M algorithm (discussed in MLMI1 and 4F10).

Note: when covariance matrix is diagonal, only need to estimate diagonal elements (variances of individual features). This can be done for each dimension of observation vectors separately.

E-M steps in Baum-Welch Estimation

Training algorithm guarantees that for each new estimate, model is more likely to generate the training data than the previous estimate, unless a **local** optimum has been reached.

$$p(\mathbf{O}|\hat{\lambda}) \geq p(\mathbf{O}|\lambda)$$

λ is the old parameter set; $\hat{\lambda}$ is the new parameter set; **O** is the training observation sequence.

By repeating this process many times a **local** maximum will be reached.

Baum-Welch estimation (i.e. EM for training HMMs) has two distinct stages.

1. **Expectation**: calculate the **posterior** probability of a feature vector being generated by a particular state. This probability (soft alignment) will be denoted as $L_j(t)$

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

   In words: the probability that the feature vector at time t was generated by state j, given the whole training sequence and the current set of model parameters.
   The combination of the *observed* data **O** and the **unobserved** data

   $$\{\{L_2(1), \ldots, L_{N-1}(1)\}, \ldots, \{L_2(T), \ldots, L_{N-1}(T)\}\}$$

   is called the **complete dataset**;
2. **Maximisation**: using the complete dataset obtain the maximum likelihood estimate of the model parameters. The exact form of the parameter estimation depends on the HMM being trained.

These two stages are repeatedly applied in training the HMMs

Auxiliary Function for HMM Training

At each iteration of E-M, a new **auxiliary function** is maximised. Define

$$\mathcal{Q}(\lambda, \hat{\lambda}) = \sum_{\mathbf{X}} P(\mathbf{X}|\mathbf{O}, \lambda) \ln p(\mathbf{O}, \mathbf{X}|\hat{\lambda})$$

Optimising $\mathcal{Q}(\lambda, \hat{\lambda}) - \mathcal{Q}(\lambda, \lambda) \geq 0$ gives a lower bound on increase in the log likelihood.
The auxiliary function can be expanded using the formula for the log likelihood of an HMM to

$$\mathcal{Q}(\lambda, \hat{\lambda}) = \sum_{\mathbf{X}} P(\mathbf{X}|\mathbf{O}, \lambda) \sum_{t=0}^{T} \ln \hat{a}_{x(t),x(t+1)} + \sum_{\mathbf{X}} P(\mathbf{X}|\mathbf{O}, \lambda) \sum_{t=1}^{T} \ln \hat{b}_{x(t)}(\mathbf{o}_t)$$

These terms can then be maximised separately.
For the transition probabilities the first term can be rewritten

$$\sum_{\mathbf{X}} P(\mathbf{X}|\mathbf{O}, \lambda) \sum_{t=0}^{T} \ln \hat{a}_{x(t),x(t+1)} = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=0}^{T} P(x(t) = i, x(t+1) = j|\mathbf{O}, \lambda) \ln \hat{a}_{ij}$$

Given stochastic constraints (& using Lagrange multipliers), this results in update equations.
For the output pdfs these can similarly maximised noting that for a Gaussian pdf

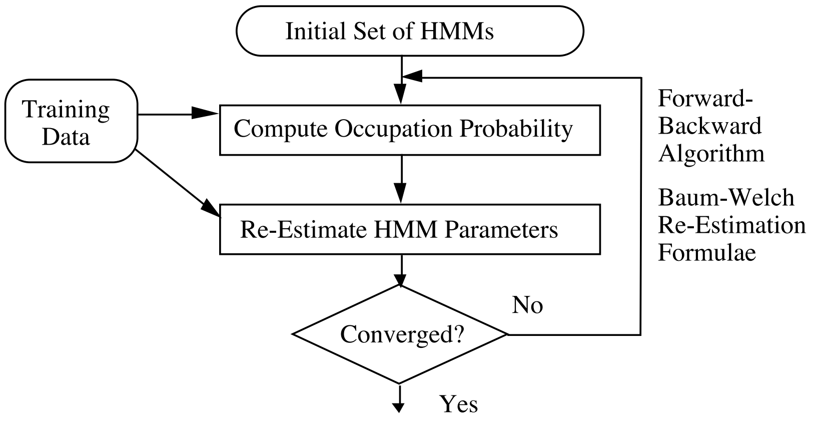
$$\sum_{\mathbf{X}} P(\mathbf{X}|\mathbf{O}, \lambda) \sum_{t=1}^{T} \ln \hat{b}_{x(t)}(\mathbf{o}_t) = \sum_{j=2}^{N-1} \sum_{t=1}^{T} P(x(t) = j|\mathbf{O}, \lambda) \ln \mathcal{N}(\boldsymbol{o}_t; \hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}_j)$$

which when maximised results in the re-estimation formulae given.

Baum-Welch Training

To estimate the model parameters the **a-posteriori state occupation probability**, $L_j(t) = P(x(t) = j|\mathbf{O}, \lambda)$ is required. This is achieved using the **forward-backward algorithm**.

As we have seen, like Viterbi training, the Baum-Welch formulae are used in an iterative fashion as shown below.


Initial Set of HMMs
Training Data
Compute Occupation Probability
Forward-Backward Algorithm
Re-Estimate HMM Parameters
Baum-Welch Re-Estimation Formulae
Converged?
No
Yes

At each iteration the likelihood is again guaranteed to increase.

Note that in both Viterbi training and Baum-Welch training, initial HMM models are needed.

Initial Model Estimates

To perform either Viterbi or Baum-Welch training, initial estimates of model parameters are required.

Possible initialisation schemes are:

- **phone-level labels**
  If the database has been labelled at the phone level then these may be used in the training. The vast majority of speech databases (TIMIT is an exception) do not have these labels. Still need state-level segmentation.
- **flat start**
  all the models are initialised with the same parameters. For Gaussian pdfs, these are typically set as the global mean and variance of all the training data.
- **best previous models**
  use the best set of models on a “similar” database as the initial estimates for the new database.

In practice flat-start or best previous models are normally used.

For a flat start to be effective, need to use the Baum-Welch algorithm. Note that this is often used for large-scale datasets (including for continuous speech training).

Forward algorithm

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

- Define the **forward** variable

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

- $\alpha_j(t)$ is the likelihood of the HMM producing all observations up to time t and occupying state j at time t:

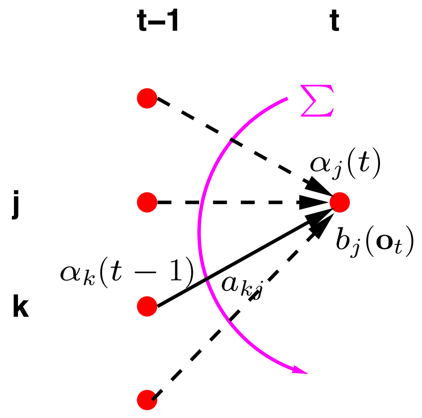
$$p(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t|\lambda) = \sum_{j=1}^{N} \alpha_j(t)$$

The likelihood of state j at time t can be found from 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 the recursive compututation of $\alpha_j(t)$:

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



Forward algorithm - Steps

The forward algorithm allows the efficient computation of the total likelihood. Given an HMM with non-emitting entry and exit states 1 and N.

Initialisation

$\alpha_1(0) = 1.0$ and $\alpha_j(0) = 0$ for $1 < j \le N$ and $\alpha_1(t) = 0$ for $1 < t \le T$

Recursion

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

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

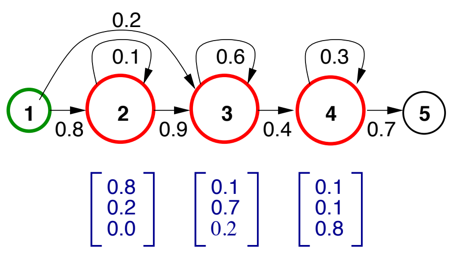
Termination

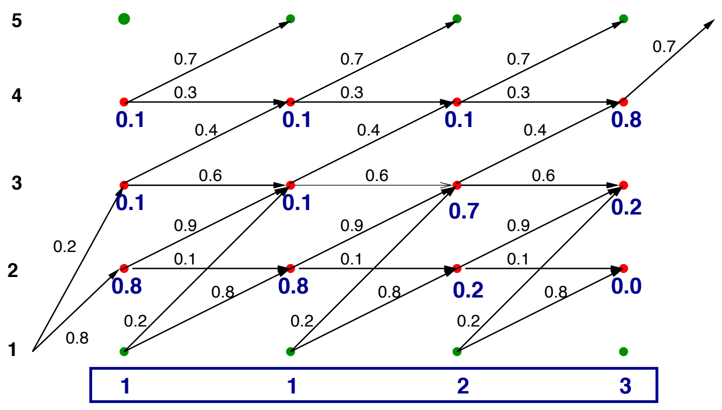
$$p(\mathbf{O}|\lambda) = \sum_{k=2}^{N-1} \alpha_k(T) a_{kN}$$

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)





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 also define the **backward variable**

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

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 note

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

Hence the required posterior probability of taking a transition between states i and j between time t and time $t+1$ is

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

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

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

Termination

$$p(\mathbf{O}|\lambda) = \beta_1(0) = \sum_{k=2}^{N-1} a_{1k} b_k(\mathbf{o}_1) \beta_k(1)$$

Baum-Welch re-estimation Formulae (Gaussian output pdfs)

For R training utterances, the update formulae are:

- **Transition probabilities**

$$\hat{a}_{ij} = \frac{\sum_{r=1}^{R} \frac{1}{p(\mathbf{O}^{(r)}|\lambda)} \sum_{t=1}^{T^{(r)}-1} \alpha_i(t) a_{ij} b_j(\mathbf{o}_{t+1}^{(r)}) \beta_j(t+1)}{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_i^{(r)}(t)} \qquad \text{for} \quad \begin{array}{l} 1 \leq i < N \\ 1 \leq j < N \end{array}$$

$$\hat{a}_{iN} = \frac{\sum_{r=1}^{R} L_i^{(r)}(T^{(r)})}{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} L_i^{(r)}(t)} \qquad \text{for} \quad 1 \leq i < N$$

- **Output distribution parameters (Gaussian)**

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

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

for $1 < j < N$

Normalisation

The magnitude of $\alpha_i(t)$ decreases at each time step. This effect is dramatic in HMMs for ASR since average values for $b_i(t)$ are normally very small and leads to **computational underflow**.

Two approaches have been used:

- Scale the $\alpha_j(t)$ at each time step so that $\sum_{j=1}^{N-1} \alpha_j(t) = 1$.
  The product of the scale factors can be used to calculate $p(\mathbf{O}|\lambda)$. Since this underflows, find $\log p(\mathbf{O}|\lambda)$ as the sum of the logs of the scale factors.
- Use a logarithmic representation of $\alpha_i(t)$. Many practical HMM systems (including HTK) use this solution. However, the forward recursion requires both multiplication and addition of log-represented numbers. Use the following method to evaluate $\log(A + B)$ when knowing $\log A$ and $\log B$:

$$\log(A+B) = \log A\left(1+\frac{B}{A}\right) = \log A + \log\left(1+\frac{B}{A}\right)$$

- The formula only needs to be evaluated if $\frac{B}{A}$ is sufficiently large
  (i.e. $\log\left(1+\frac{B}{A}\right)$ sufficiently > 0).
  Fast implementation: look-up table of $\log\left(1+\frac{B}{A}\right)$ against $\log\frac{B}{A} = \log B - \log A$.

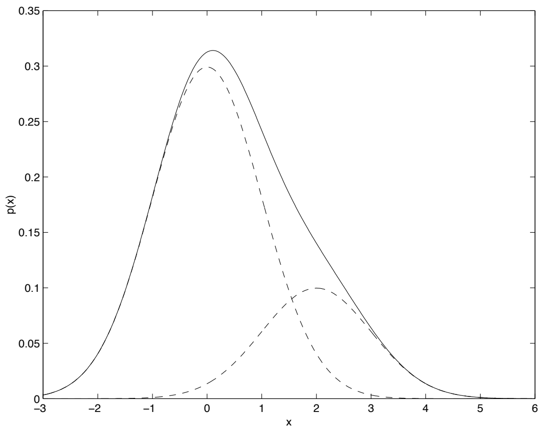
Gaussian Mixture Output Distributions

A single (multivariate) Gaussian to model the output distribution may be poor. A better & more general model may be a **mixture of Gaussians**:

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

c_{jm} is the component prior (or weight). For this to be a pdf it is necessary that

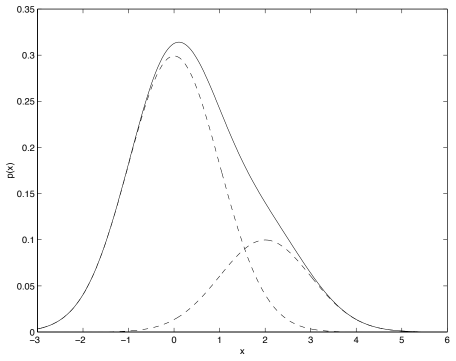
$$\sum_{m=1}^{M} c_{jm} = 1 \quad \text{and} \quad c_{jm} \geq 0$$

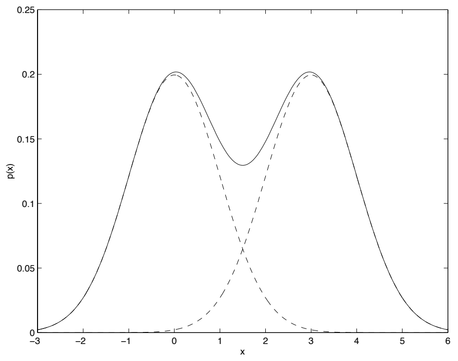


- A two component Gaussian mixture distribution is shown.
- Using Gaussian mixtures it is possible to approximate any distribution (provided you have enough components).
- Mixture distributions allow very flexible modelling of the acoustic vectors associated with a state.

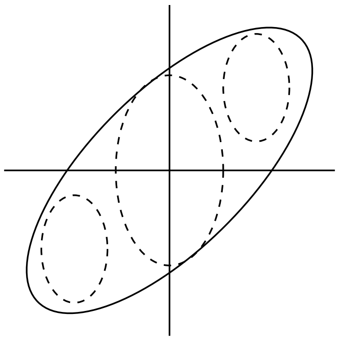
Modelling PDFs

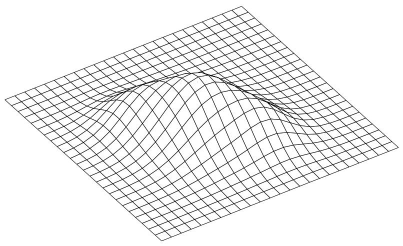
- **Asymmetric** and **Bimodal** distributions





- **Correlation** Modelling





Model Parameters for HMMs with GMM Output PDFs

The parameters that need to be stored are

1. the transition matrix (standard HMM);
2. for *each* state the set of component priors (weights), means and variances

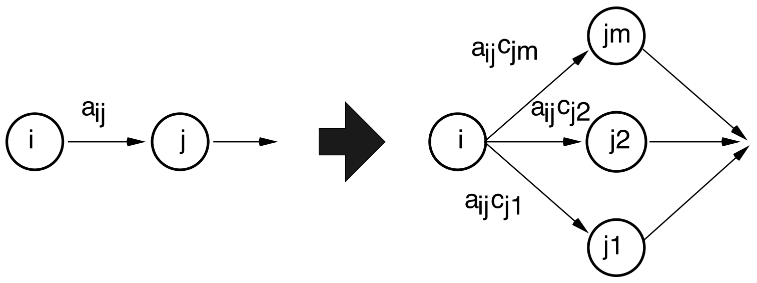
$$\{\{c_{j1}, \boldsymbol{\mu}_{j1}, \boldsymbol{\Sigma}_{j1}\}, \ldots, \{c_{jM}, \boldsymbol{\mu}_{jM}, \boldsymbol{\Sigma}_{jM}\}\}$$

Note: multiple component distributions is that it may result in a large number of Gaussians, hence system parameters.

Contrast parameters (observation dimensionality $d = 39, M = 10$):

- **Single Full Covariance Gaussian**:
  mean requires d parameters, covariance matrix $\frac{d(d+1)}{2}$ - 819 parameters.
- **Single Diagonal Covariance Gaussian**:
  mean requires d parameters, covariance matrix d - 78 parameters.
- **Multiple Diagonal Covariance Gaussian Components**:
  M components require Md parameters for the mean Md parameters for the diagonal variances and $M - 1$ for weights - 789 parameters.

For Baum-Welch re-estimation each of the component Gaussians may be considered as a separate state thus



The alignment for a frame is to a particular Gaussian component of a particular state. Thus

$$\begin{aligned} L_{jm}(t) &= P(x(t) = jm|\mathbf{O}, \lambda) \\ &= \frac{1}{p(\mathbf{O}|\lambda)} \sum_{i=2}^{N-1} \alpha_i(t-1) a_{ij} c_{jm} b_{jm}(\mathbf{o}_t) \beta_j(t) \end{aligned}$$

The estimates of the mean and variance will be the similar to the single Gaussian case for a single training utterance

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{t=1}^{T} L_{jm}(t)\mathbf{o}_t}{\sum_{t=1}^{T} L_{jm}(t)}$$

It is also necessary to estimate mixture component priors. Similar to transition probabilities, taking account of the stochastic constraints, mixture component priors can be found as

$$\hat{c}_{jm} = \frac{\text{Estimated Number of vectors from component } m \text{ state } j}{\text{Estimated number of vectors from state } j}$$

In terms of the posterior probability of state component occupation and state occupation this becomes for a single training utterance

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

The extension to multiple training utterances is straightforward.

Viewing Gaussian mixture components as parallel states is important and widely used in estimation of both mixture components themselves and other parameters estimated for GMM-HMM based systems (such as for adaptation).

Training Gaussian Mixture HMMs

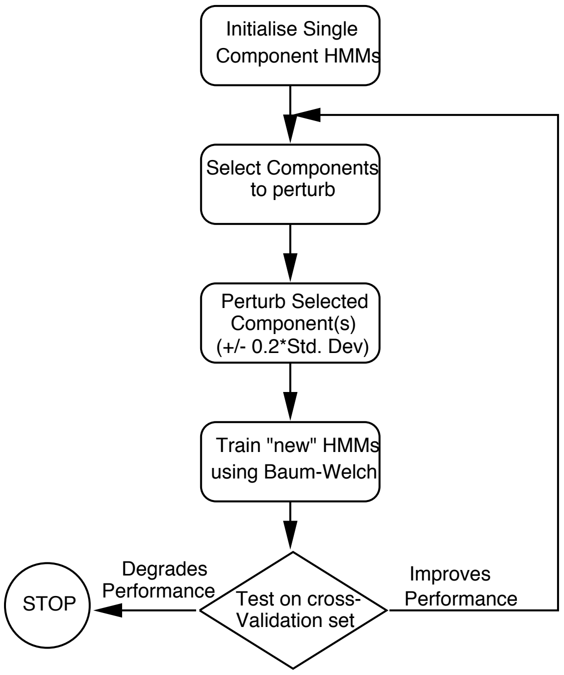
It is, again, necessary to have initial estimates of the models.

If using *best previous set* no problem. However, not possible to use flat start.

Two options:

1. **Clustering**. Gather together all aligned vectors for a state and used clustering (or VQ) techniques to initialise a set of Gaussians (need to segment data to do this).
2. **"Mixing-Up"**. Iterative routine as shown on right.

   Use of cross validation set allows "sensible" stop criteria.

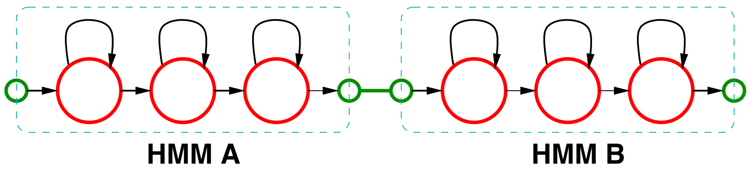

Initialise Single Component HMMs
Select Components to perturb
Perturb Selected Component(s) (+/- 0.2*Std. Dev)
Train "new" HMMs using Baum-Welch
Test on cross-Validation set
Degrades Performance
STOP
Improves Performance

Training for Continuous Speech Recognition

There are a number of ways to obtain suitable models for continuous speech recognition.

(i) **Isolated Word Training**
Use HMMs trained on isolated words, does not reflect additional variability

(ii) **Segmentation of continuous utterances**
Find the segment boundaries using Viterbi alignment, recording the boundary time of each word/model boundary. This runs Viterbi on a composite model of each training utterance.



Train on the resulting segmented data. Note that segmentation and parameter re-estimation can be iterative.

(iii) **Sentence-level Baum-Welch training**
For each utterance, construct a composite HMM model for that utterance by concatenating the HMMs in sequence.

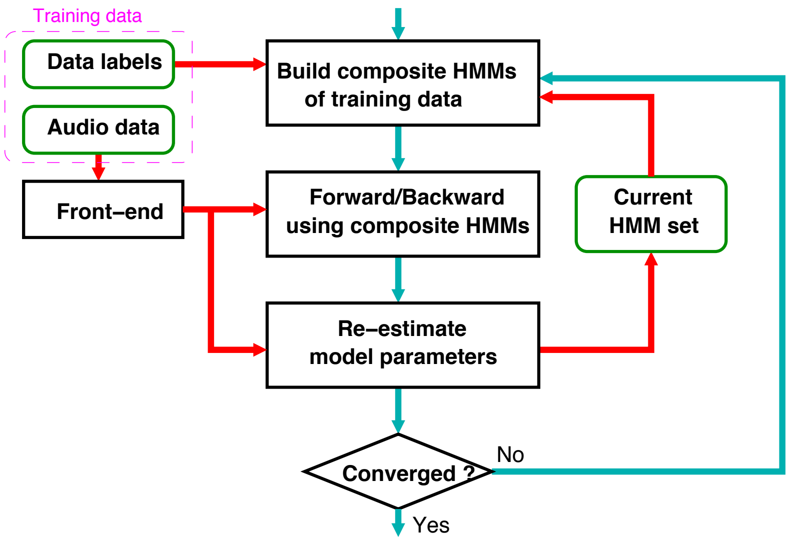
Run the F-B algorithm and then accumulate statistics for all HMMs at once. Uses "probabilistic segmentation", soft decisions on model boundaries.

Initialise models from (i) or (ii) for word HMMs (or use flat-start approach).

Sentence-Level Baum-Welch Training

In addition to initial HMMs and the the audio data, a known model sequence is required. If only word-level training transcripts are available, for sub-word unit training, the model sequence will come from a pronunciation dictionary: more on this later.

Note that a flat start, i.e. all models with equal arbitrary initial model parameters, is possible! This relies on only the model sequence without any further information on model boundaries.


Training data
Data labels
Audio data
Front–end
Build composite HMMs of training data
Forward/Backward using composite HMMs
Re–estimate model parameters
Current HMM set
Converged ?
No
Yes

Summary

- Described **Baum-Welch** re-estimation of continuous density HMMs taking into account all possible state sequences.
- Alternative HMM maximum-likelihood scheme to Viterbi training.
- Instance of the Expectation-Maximisation (E-M) algorithm (developed for HMMs before general case).
- Efficient recursions for generating the **forward** and **backward** probabilities.
- Forward and backward values combined to find **posterior probability of state occupation**.
- Baum-Welch re-estimation steps need to be applied iteratively.
- Models must be initialised appropriately.
- Baum-Welch re-estimation allows the use of a flat-start for model initialisation.
- Extended the output distribution modelling from single Gaussian distributions to **Gaussian mixtures**. The extensions to the Baum-Welch re-estimation formulae were also given.
- HMMs can be trained for continuous speech, including by sentence-level Baum-Welch (embedded training).