Engineering Part IIB: Module 4F11
Speech and Language Processing
Lectures 4/5 : Speech Recognition Basics

Phil Woodland: pcw@eng.cam.ac.uk

Lent 2016

Cambridge University Engineering Department

What is Speech Recognition?

Automatic Speech Recognition (ASR) converts an unknown speech waveform into the corresponding orthographic transcription: normally a **string of words**.

Speech recognition is difficult because of

- differences between speakers **inter-speaker** variability;
- differences in how a speaker utters the same word **intra-speaker** variability;
- acoustic channel & noise - microphone differences, background noise;
- need to model language in general and of particular domain;

From the word strings other "downstream" applications may be used e.g.

1. machine-human dialogue systems
2. machine translation
3. information retrieval and information extraction

Example Applications of ASR

- **Desktop dictation**: “voice typewriter”
- **Telephony & Information Access systems**:
  - banking, account access and management
  - brokerage, share quotes and trading
  - voice access to web pages/services & voice search
- **Transcription Systems**
  - Often specialised domains: legal, medical, voicemail etc
  - Transcription for information retrieval
- **Command and control**:
  - navigation around computer windows system etc.
  - control of many home/office/military items
  - Voice dialling on mobile phones (hands/eyes busy elsewhere)

Applications has been rapidly growing as technology/compute power improves.

Task Complexity

There are several styles of speech recognition

	input mode	vocab size	basic unit	grammar
isolated	discrete	small	word	none
continuous	continuous	medium	phone	FS / N-gram
discrete LV	discrete	large	phone	N-gram
cont. LV	continuous	large	phone	N-gram

- **input mode**: discrete (gaps between words) vs. continuous speech
- **vocab. size**: small (< 200 words) to very large (> 60k words)
- **basic unit**: *acoustic model* units used (discussed later)
- **grammar**: assigns a probability to possible word sequences
  - finite state networks only allow paths through an explicit word network;
  - N-gram grammars allows all word sequences with non-zero probability,

Also operating condition affects performance and task complexity.

- **Speaker mode**: speaker dependent vs speaker independent
- **Microphone**: close vs far-field; fixed vs variable; bandwidth
- **Background noise**: clean vs noisy (& noise type)
- **Channel**: high quality/std telephone/cellular
- **Speaking style**: prepared/read/careful vs spontaneous/casual

All modern ASR systems are based on **statistical pattern recognition**.

The next few lectures will

1. explain basic statistical pattern recognition principles in the context of *isolated word recognition*.
2. extend these concepts to *large vocabulary continuous speech recognition*

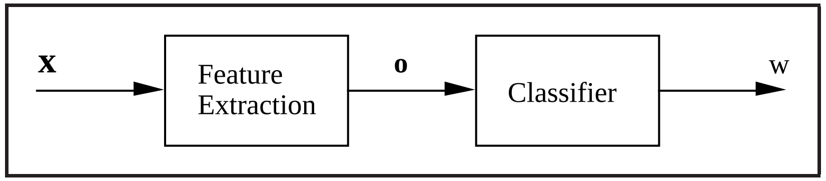
Pattern Recognition Framework

A set of features $\mathbf{o}$ is derived from a measurement $\mathbf{x}$ and a class ω is identified by finding the most likely class ω given the data $\mathbf{o}$, i.e.

$$\hat{\omega} = \arg\max_{\omega}\{P(\omega|\mathbf{o})\}$$

$P(\omega|\mathbf{o})$ is unknown so Bayes' rule is used: $P(\omega|\mathbf{o}) = \frac{p(\mathbf{o}|\omega)P(\omega)}{p(\mathbf{o})}$ where $p(\mathbf{o}|\omega)$ is the **likelihood** of the data given the class (defined by a model), $P(\omega)$ is the prior probability of the class (defined by a model). Since the maximisation does not depend on $p(\mathbf{o})$, then

$$\hat{\omega} = \arg\max_{\omega}\{p(\mathbf{o}|\omega)P(\omega)\}$$



Statistical Speech Recognition

In ASR the task is to find the most likely word sequence $\hat{W}$ from an utterance $\mathbf{O}$.

$$\hat{W} = \arg\max_{W} \{p(\mathbf{O}|W)P(W)\}$$

The essential parts of an ASR system are

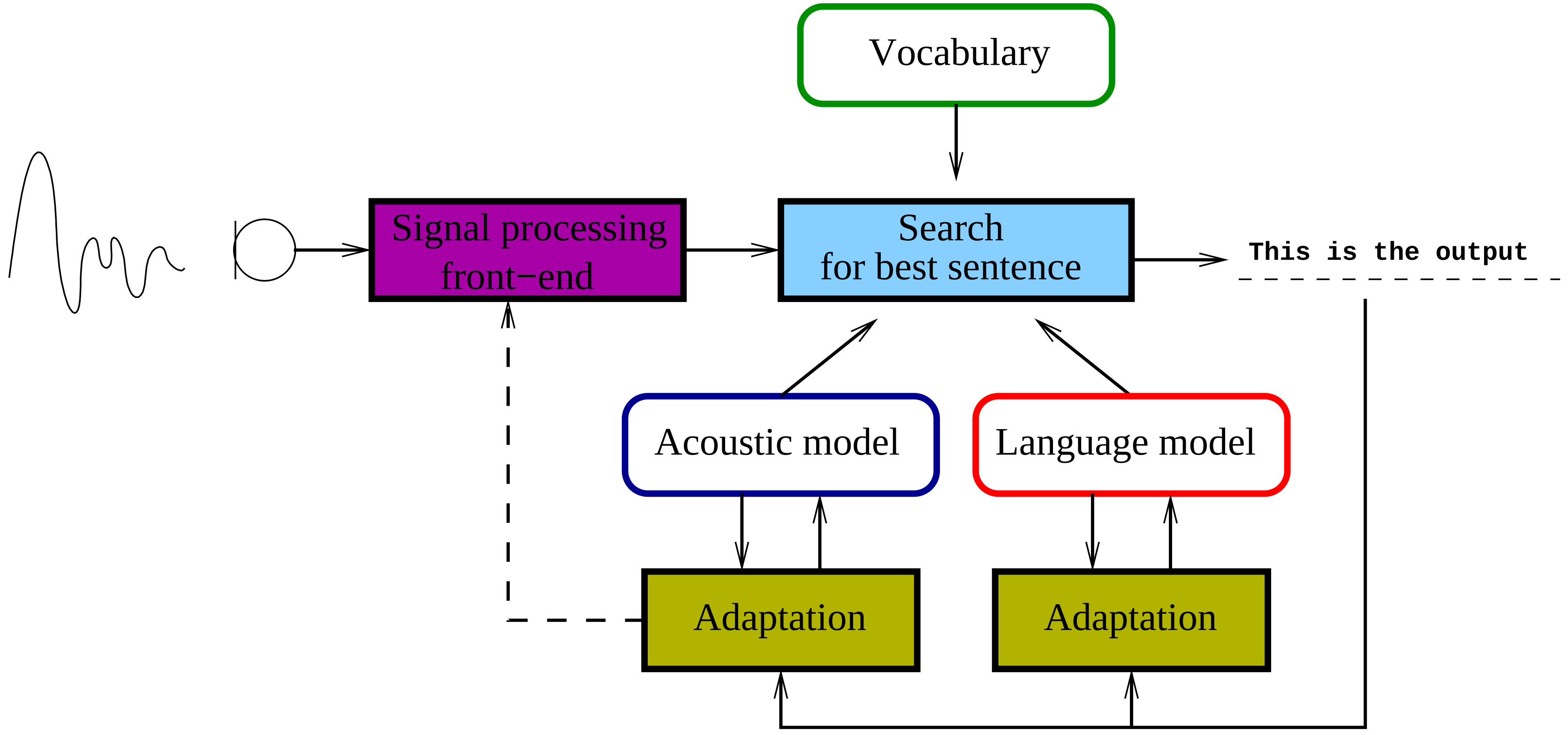
- **acoustic model** giving $p(\mathbf{O}|W)$
- **language model** yielding $P(W)$

Hence, the ASR problem depends on finding solutions to two problems

- **training** - finding a suitable representation for the acoustic and language models
- **recognition** - finding the most likely word sequence

The most common form for the acoustic model is a **Hidden Markov Model (HMM)**.

Generic Recognition Architecture



A search is performed for the most likely word or sentence given the acoustic and language models (recognition,decoding).

A finite set of words is defined in the vocabulary of the ASR system.

- The acoustic models usually make use of phonetic representation of words (although models for individual words can also be used and are considered first).
  - Also need to supply the pronunciations for each word
- Words not in the vocabulary (Out-Of-Vocabulary) cause errors.
- In some cases no (a null) language model is appropriate - otherwise normally an N-gram is used.
- The output of the ASR system may be used to adjust the models (acoustic or language model adaptation: beyond the scope of this course).

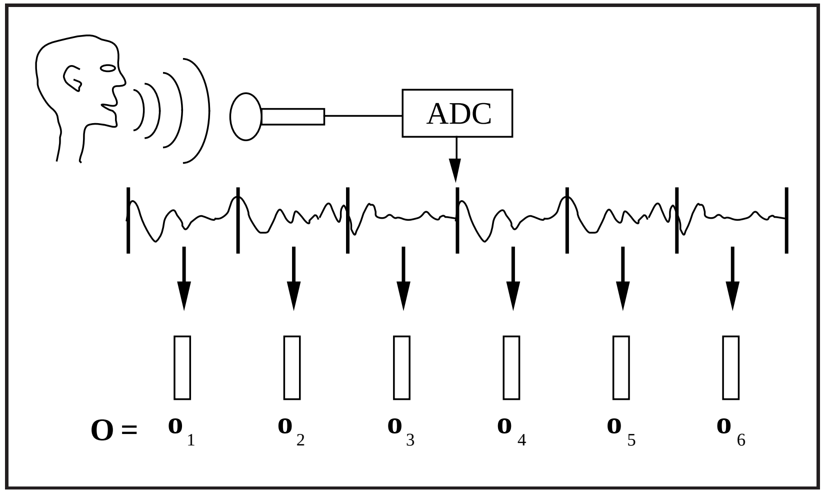
Isolated word recognition simplifies the system by pre-segmentation of the speech data. However, this segmentation is not straightforward in anything but a low-noise audio environment.

Feature Extraction

The aim of the feature extraction is to

- transform raw-data to a form suitable for the classifier;
- reduce the data rate;
- keep only information that discriminates between classes

Since speech signal is not fixed length, use multiple fixed duration feature vectors.


ADC
$\mathbf{O} = \mathbf{o}_1 \; \mathbf{o}_2 \; \mathbf{o}_3 \; \mathbf{o}_4 \; \mathbf{o}_5 \; \mathbf{o}_6$

Normally in ASR a fixed duration of speech signal (a frame) is used for each feature vector and consists of information describing the short-term spectrum, which is often encoded as a cepstral representation such as MFCCs.

Hidden Markov Models

A classifier is needed that can handles a variable number of feature vectors from each speech unit. The standard model is the Hidden Markov Model (HMM).

Assumptions:

1. The features (observations) accurately represent the signal. Speech is assumed to be stationary over the length of the frame. Frames are usually around 25msecs, so for many speech sounds this is not a bad assumption.
2. Observations are independent given the state that generated it. Previous and following observations do not affect the likelihood. This is not true for speech, speech has a high degree of continuity.
3. Between state transition probabilities are constant. The probability of from one state to another is independent of the observations and previously visited states. This is not a good model for speech.

Despite its limitations HMMs are the most widely used and, to date, successful acoustic models for speech recognition.

HMMs (cont)

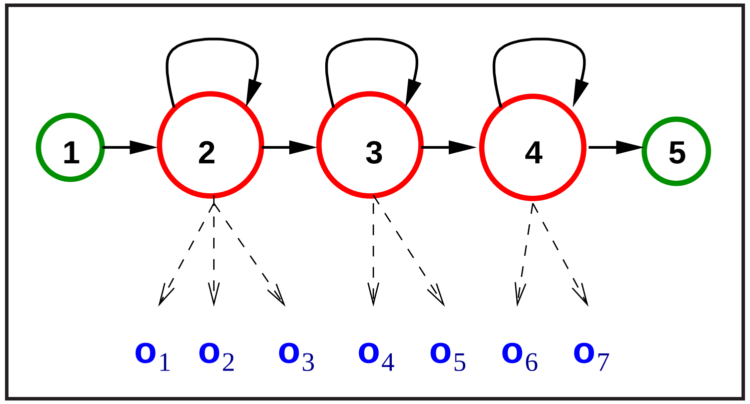
An HMM is a finite state machine:

1. N states with $N-2$ **emitting states**.
2. it has a non-emitting **entry** state and a non-emitting **exit** state (some formulations don't explicitly include these).
3. any pair of states i and j can be connected by a transition with probability a_{ij}
4. it changes from current state i to new state j with probability a_{ij} every input frame
5. when an emitting state j is entered, acoustic feature vector $\mathbf{o}$ is generated with probability $b_j(\mathbf{o})$

The HMM, λ, may be written as $\lambda = \{N, \{a_{ij}\}, \{b_j(\cdot)\}\}$

Note: the HMM is a *generative* model of speech. It is used to find the likelihood that the model generated the observed data.

Example HMM


1
2
3
4
5
$\mathbf{o}_1$ $\mathbf{o}_2$ $\mathbf{o}_3$ $\mathbf{o}_4$ $\mathbf{o}_5$ $\mathbf{o}_6$ $\mathbf{o}_7$

Let

- $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2 \ldots \mathbf{o}_T\}$ be the observed data
- $\mathrm{X} = \{x(1), x(2), \ldots, x(T)\}$ be the specified state sequence

For speech recognition, normally HMMs are **left-to-right**.
The joint probability is

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

where

- $x(0)$ is always the entry state 1
- $x(T+1)$ is always the exit state N.

Likelihood Calculation

Using the HMM from the previous slide

$$p(\mathbf{O},\mathbf{X}|\lambda) = a_{12}b_2(\mathbf{o}_1)a_{22}b_2(\mathbf{o}_2)a_{22}b_2(\mathbf{o}_3)a_{23}\ldots b_4(\mathbf{o}_7)a_{45}$$

We have assumed a state sequence and hence know the state that generated each observation vector. However for actual speech the state sequence is not known known! It is **hidden**!

Therefore to compute the likelihood for an HMM given just the observation vectors, need to sum over all possible state sequences

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

Evaluating this expression directly is impractical since there are too many state sequences. However, as will be shown later, there are recursive algorithms which allow $p(\mathbf{O}|\lambda)$ to be computed efficiently while effectively summing over all state sequences.

Output Distributions

The HMM output distribution should be chosen to:

- closely match the actual distribution of the data associated with a state
- be mathematically and computationally tractable

A common simple choice is the multivariate Gaussian (for feature vectors)

$$b_j(\boldsymbol{o}) = \mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

where

$$\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{o}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\boldsymbol{o}-\boldsymbol{\mu})}$$

Model parameters:

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

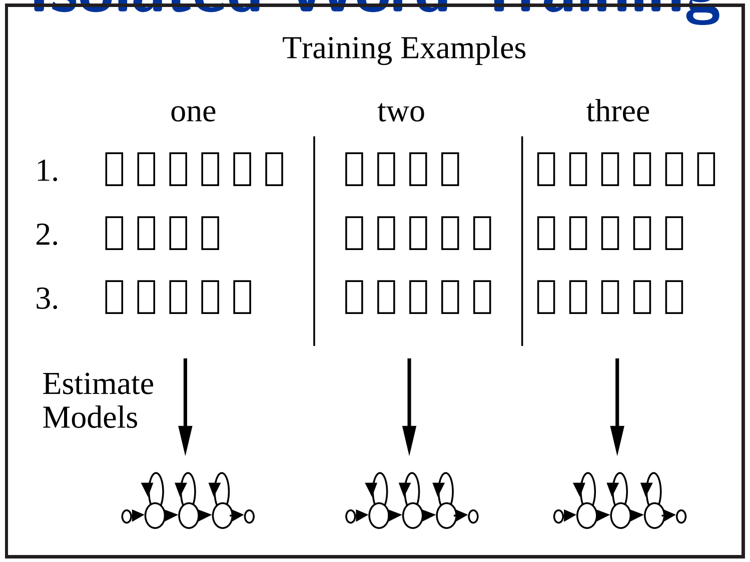
Not ideal for spectral vectors as:

- speech power spectra are not Gaussian
- elements of the feature vector highly **correlated**, would require a full covariance matrix.

Cannot directly use short-term power spectrum. Log spectrum is more Gaussian and if use MFCCs also allow the use of diagonal covariance matrices (far fewer parameters).

Note that the HMMs used here for speech recognition are **Continuous Density** HMMs. An alternative that is used for symbolic data are **discrete density** HMMs in which the output probability is from a discrete distribution over the set of observation symbols i.e. the probability of each possible symbol occurring. We will use these in some examples for convenience/compactness. They can be used for speech if a process of **vector quantisation** is employed.

Isolated Word Training



Initially use a separate HMM for each vocabulary word. Then

1. select HMM topology i.e. the number of states and the connectivity
2. record a number of spoken examples of that word (or use a database)
3. the parameters of the HMM are chosen to maximise the total likelihood of all the training examples of that word (maximum likelihood training).

Need a maximum-likelihood estimation scheme to obtain the model parameters.

Isolated word recognition

If individual words are spoken in **isolation**, can reduce the complexity of the search for the best word sequence.

Assume words are separated from each other (in a sentence) by detectable pauses. The task for the acoustic model is simplified to provide an estimate of likelihood for each individual word. The front-end produces a stream of feature vectors (observation vectors)

For isolated word recognition, assume that we have a set of pre-trained HMMs for each vocabulary word. If no grammar is used then process is

1. parameterise unknown speech to give a sequence $\mathbf{O}$
2. for each HMM word model λ_i compute the likelihood of generating the sequence, $p(\mathbf{O}|\lambda_i)$
3. Assuming equal prior probabilities, the HMM with the highest likelihood identifies the word

Note that for both training and recognition need a way to rapidly compute the likelihood that a model generated a sequence of observed speech vectors.

ASR Introduction Summary

- A particular instance of a speech unit (phone, word, sentence etc) will consist of a variable length sequence of spectral vectors.
- Each spectral vector has variability due to speaker and environment.
- An HMM is a statistical model which represents variability in duration through its state transition structure $\{a_{ij}\}$ and variability in spectra through its output distributions $\{b_j(\cdot)\}$.
- For isolated word recognition with word-based HMMs, the parameters are estimated directly from training data consisting of spoken examples of each word to be recognised.
- For isolated word recognition with equal priors, words are recognised by finding the word HMM which yields the highest likelihood for the speech data.

Maximum Likelihood Estimation for HMMs

Initially HMM training will be considered, and standard **maximum likelihood** parameter estimation will be described. Here we need to find the HMM model parameters, $\hat{\lambda}$, so that

$$\hat{\lambda} = \arg\max_{\lambda} \{p(\mathbf{O}|\lambda)\}$$

where $\mathbf{O}$ is the training data. Initially we will assume that there is just a single training utterance for the HMM with T frames, i.e. that

$$\mathbf{O} = \boldsymbol{o}_1, \ldots, \boldsymbol{o}_T$$

Other forms of training are possible. In state-of-the-art HMM-based speech recognition systems *discriminative* training techniques are starting to be commonly used (beyond the scope of this course).

Parameter Estimation for 1-state HMM (Gaussian)

Single-emitting state HMM - same, known, state throughout the data.

Gaussian parameters can be estimated as:

- **Mean**:

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

- **Covariance matrix**:

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

These are the *maximum likelihood* estimates.

Note: a single training sequence is being assumed. In practice for multiple training sequences need to sum over all observations of all training sequences.

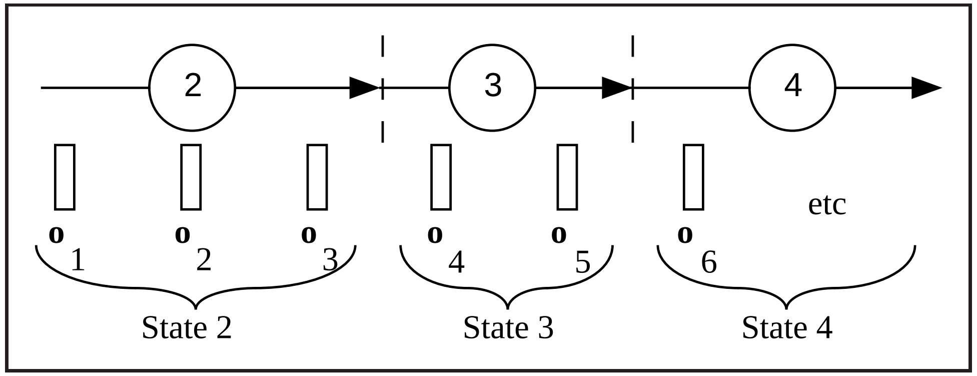
Viterbi Parameter Estimation

How to extend to a multi-state HMM?

From previous example, if we had both

- the observed data $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T$
- the most likely segmentation of the data which gives which state generated each observation vector.

then same formulae as before may be used.



- state j generates observations starting at time t_j.
- training data for state j consists of $\boldsymbol{o}_{t_j}, \boldsymbol{o}_{t_j+1} \ldots \boldsymbol{o}_{t_{j+1}-1}$.

This is the *Viterbi Segmentation* - given this segmentation, the parameters of state j can be estimated

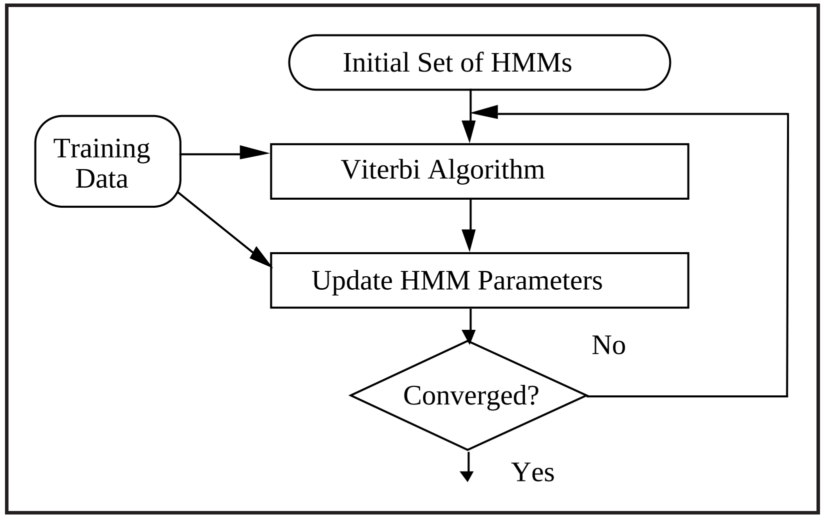
$$\hat{\boldsymbol{\mu}}_j = \frac{1}{t_{j+1} - t_j} \sum_{t=t_j}^{t_{j+1}-1} \boldsymbol{o}_t$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{1}{t_{j+1} - t_j} \sum_{t=t_j}^{t_{j+1}-1} \left[(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_j)'\right]$$

This form assumes a left-to-right structure i.e. the HMM never returns to a state once it has left. (Viterbi training can still be used in other cases - the equations need to be reformulated).

Iterative Training With the Best State-Sequence

Fortunately an efficient algorithm, the *Viterbi Algorithm*, exists to find the most likely state sequence given some model parameters.



Suggests an iterative approach, which **interleaves**

- finding best state sequence
- model parameter update

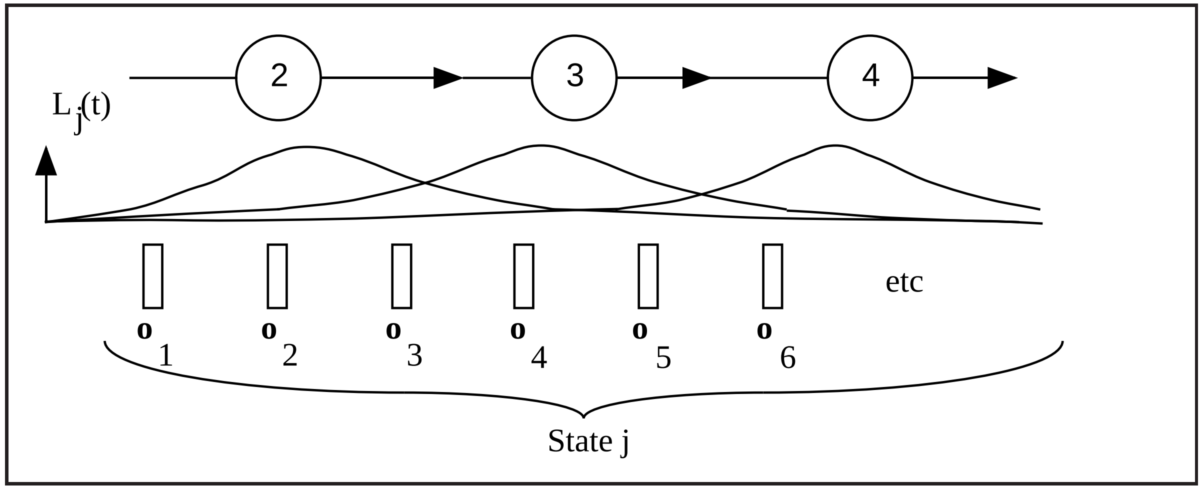
The likelihood is guaranteed to increase (or stay the same) with each iteration.

Schemes to initialise models:

- **Flat start**: all model parameters are set to be the same.
- **Hand-segment data**: an expert hand-segments a subset of the data to bootstrap the models.
- **Previous models**: models trained on another data-set are used.

Baum-Welch Parameter Estimation

Viterbi training requires a hard decision to be made about the location of state boundaries.



Alternatively:

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

Parameters can now be estimated by using weighted averages.

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

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

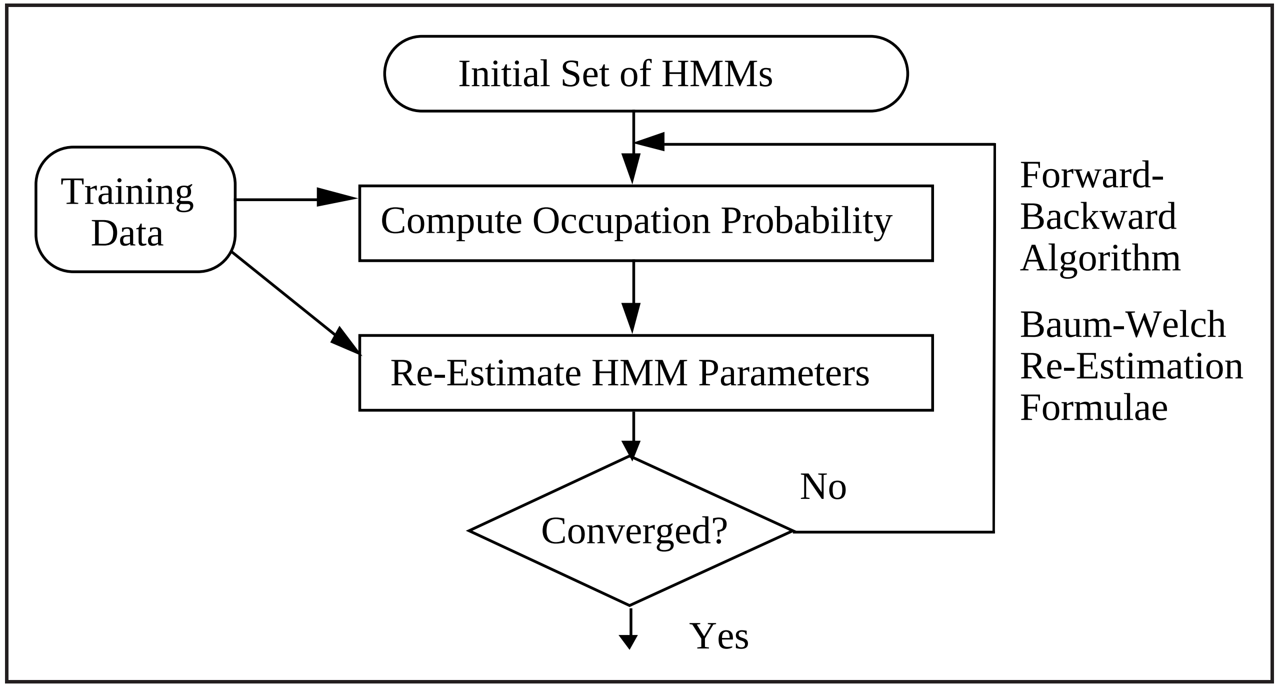
These equations are called the *Baum-Welch Re-Estimation Formulae* and are an example of the E-M algorithm (discussed in 4F10).

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

Baum-Welch Training

To estimate the model parameters the *a-posteriori* state occupation probability, $L_j(t)$ is required. This is achieved using the forward-backward algorithm.

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
Re-Estimate HMM Parameters
Converged?
No
Yes
Forward-Backward Algorithm
Baum-Welch Re-Estimation Formulae

At each iteration the likelihood is again guaranteed to increase. Models are initialised in the same fashion as for Viterbi training.

Forward algorithm

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

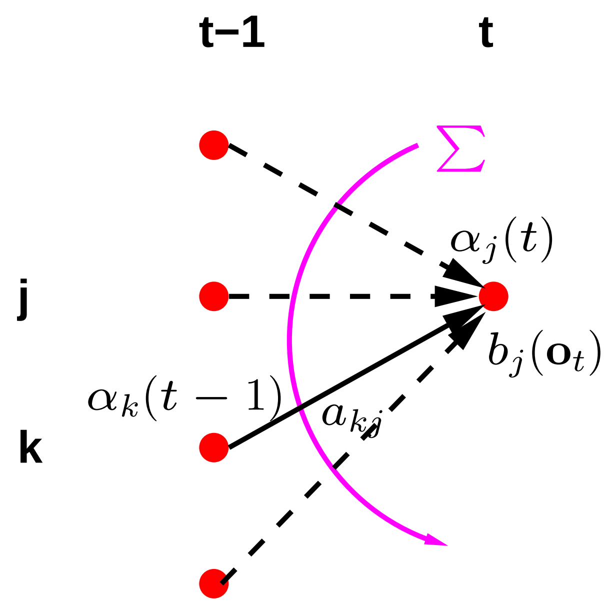
- Define the **forward** variable $$\alpha_j(t) = p(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t, x(t) = j\,|\lambda)$$
- $\alpha_j(t)$ is the likelihood of the HMM producing all observations up to time t and occupying state j at time t: $$p(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t|\lambda) = \sum_{j=1}^{N} \alpha_j(t)$$

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 $1 < j \leq N$ and $\alpha_1(t) = 0$ for $1 < t \leq T$

Recursion

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

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

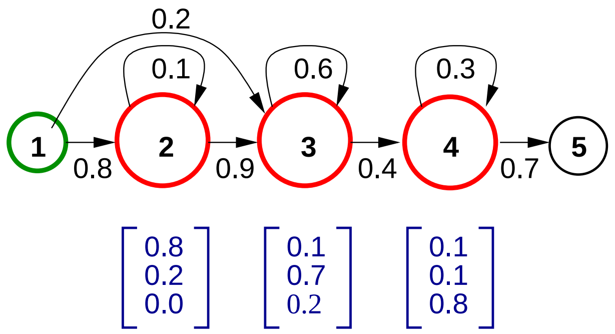
Termination

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

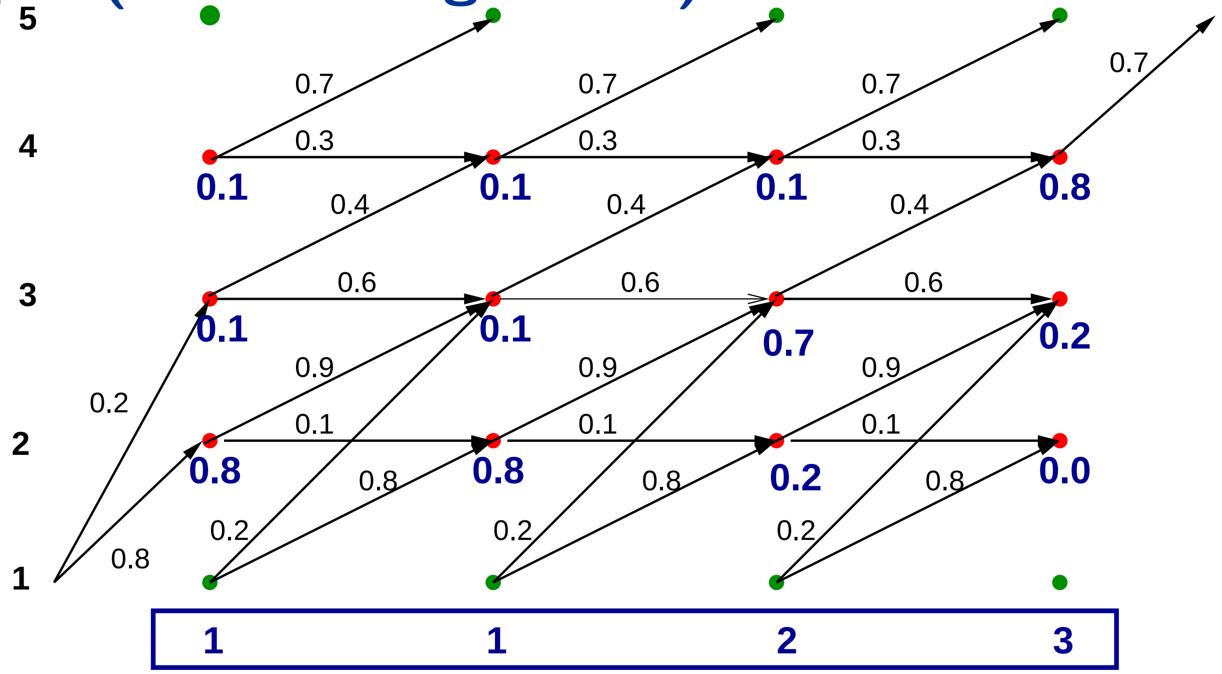
The forward algorithm can be used for recognition (at least for simple tasks):

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

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**

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

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

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

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

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

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**

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

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

- **Output distribution parameters (Gaussian)**

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

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

$$\text{for} \quad 1 < j < N$$

Normalisation

The magnitude of $\alpha_i(t)$ decreases at each time step. This effect is dramatic in HMMs for ASR since the average values for $b_i(t)$ are 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 $\log B$:

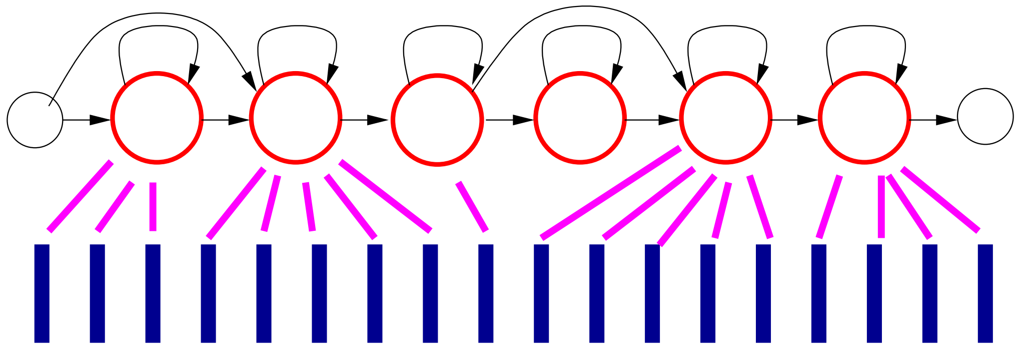
$$\log\left(A+B\right) = \log A\left(1+\frac{B}{A}\right) = \log A + \log\left(1+\frac{B}{A}\right)$$

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

The Best State Sequence

For recognition, the probability of an unknown word is required for each HMM, i.e. $p(\mathbf{O}|\lambda_i)$ is required for each model λ_i. This could be calculated using the forward algorithm as above (or equally, the backward algorithm could be used). However, in practice, an estimate of $p(\mathbf{O}|\lambda_i)$ based on just the **most likely state sequence** is preferred since it is more easily extended to handle continuous speech. The best state sequence is also useful for a number of other purposes including

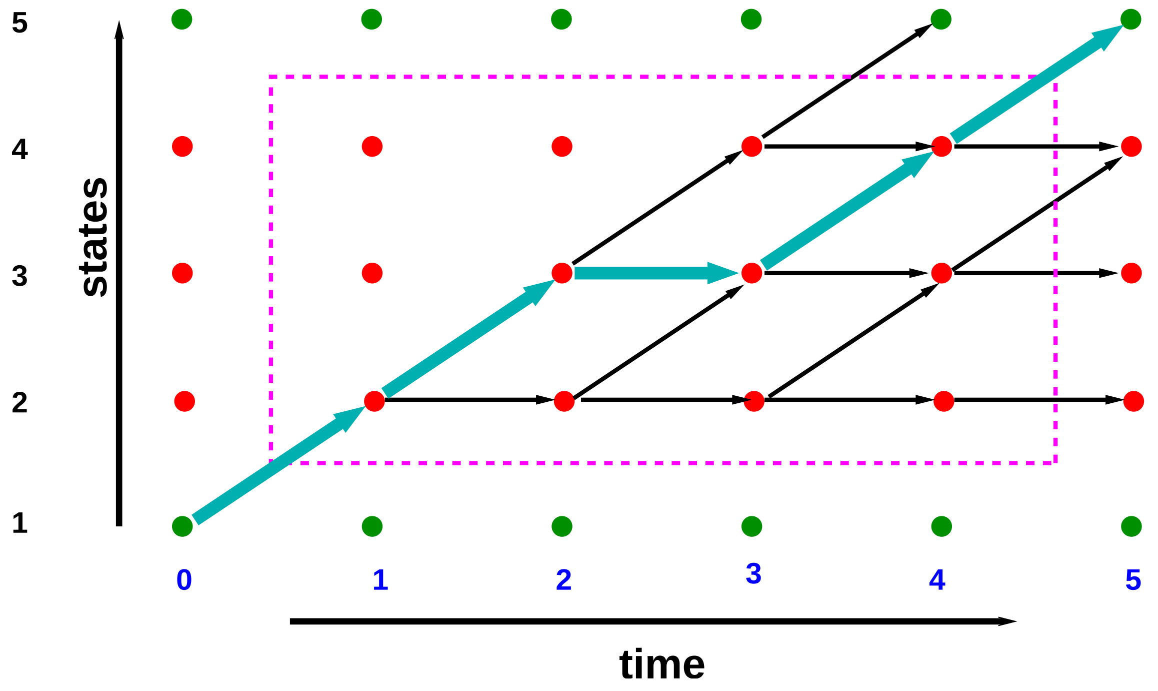
- segmenting words into smaller units (sub-word units).
- segment sentences into words
- use in Viterbi training



The Trellis

The most likely state sequence can be visualised as a **path** through the **trellis** of possible HMM paths.

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 **complete 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 a given model λ and an observation sequence $\mathbf{O}$ of length T. It computes

$$\hat{p}(\mathbf{O}|\lambda_i) = \max_{X}\{p(\mathbf{O}, X|\lambda_i)\}$$

and the associated most likely state sequence $\mathbf{X}^*$ in the maximisation.

This will involve a simple recursion similar to the computation if $\alpha_i(t)$, but rather than summing over all paths (state-sequences), the Viterbi algorithm only takes into account the most likely state sequence through the model to any point.

Let (compare with $\alpha_j(t)$)

$$\phi_j(t) = \max_{X^{(t-1)}}\{p(\boldsymbol{o}_1 \ldots \boldsymbol{o}_t, x(t) = j|\lambda_i)\}$$

where $X^{(t-1)}$ is the set of all partial paths of length $t-1$.

In words $\phi_j(t)$ represents the probability of the "best" partial path of length t through the trellis of possible state sequences ending in state j.

The partial path probability $\phi_j(t)$ can be calculated using a recursion in an exactly analogous way to the forward probability

$$\phi_j(t) = \max_i \{\phi_i(t-1) a_{ij} b_j(\boldsymbol{o}_t)\}$$

where $\phi_j(0) = 1$ if $j = 1$ and 0 otherwise.

Extending this recursion through the utterance leads to the likelihood of the best complete path. To also find the most-likely state sequence, it is necessary to store the local decisions made at each point in the Viterbi trellis and then **traceback** along the most likely path at the end of the utterance.

The Viterbi algorithm (2)

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 $\phi_1(t) = 0.0 \quad \text{for} \quad 1 \le t \le T$

Recursion

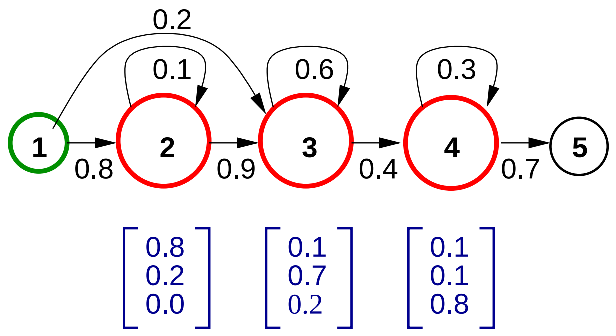
```
for $t = 1, 2, \ldots, T$
. . . for $j = 2, 3, \ldots, N-1$
. . . . . . . compute $\phi_j(t) = \max_{1 \le k < N} [\phi_k(t-1)a_{kj}]\, b_j(\mathbf{o}_t)$
. . . . . . . store the predecessor node: $\text{pred}_k(t)$
```

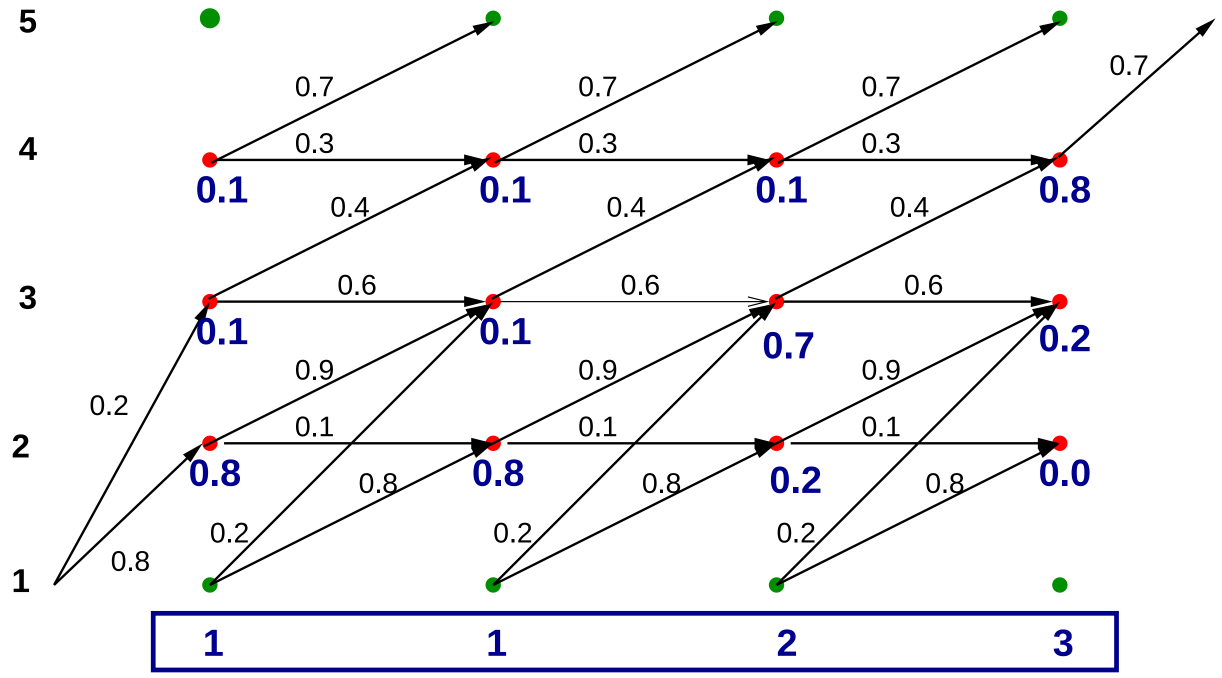
Termination

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

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

Viterbi - Example


0.2
0.1
0.6
0.3
1
2
3
4
5
0.8
0.9
0.4
0.7
$\begin{bmatrix}0.8\\0.2\\0.0\end{bmatrix}$
$\begin{bmatrix}0.1\\0.7\\0.2\end{bmatrix}$
$\begin{bmatrix}0.1\\0.1\\0.8\end{bmatrix}$



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

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

Note that the HMM assumptions are key to the efficiency of the Viterbi (and forward) algorithms.

Adding Dynamic Information

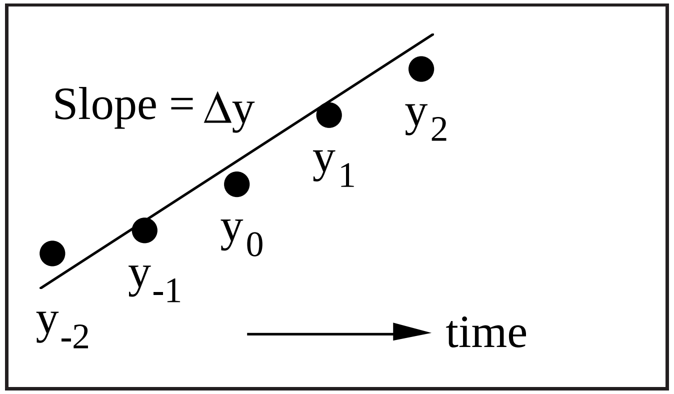
The formulation of the HMM assumes:

- each feature vector $\boldsymbol{o}_t$ is independent of all preceding and following vectors given the state that generated it.

Since this is not true for speech, this deficiency in HMM modelling can be partly overcome by modifying the fecture vectors used.

Add $delta$ coefficients to the feature vector as follows (where $\boldsymbol{y}_t$ are e.g. MFCCs)

$$\boldsymbol{o}_t = \left[\begin{array}{c} \boldsymbol{y}_t \\ \boldsymbol{\Delta y}_t \end{array} \right]$$



Each delta is a differential computed using the standard regression formula

$$\boldsymbol{\Delta y}_t = \frac{\sum_{\tau=1}^{D} \tau(\boldsymbol{y}_{t+\tau} - \boldsymbol{y}_{t-\tau})}{2\sum_{\tau=1}^{D} \tau^2}$$

where D determines the size of the *delta window* and the differential is taken as the best straight line through this window.

Many recognisers also add delta-delta parameters.

$$\boldsymbol{o}_t = \begin{bmatrix} \boldsymbol{y}_t \\ \boldsymbol{\Delta y}_t \\ \boldsymbol{\Delta^2 y}_t \end{bmatrix}$$

where

$$\boldsymbol{\Delta^2 y}_t = \frac{\sum_{\tau=1}^{D} \tau(\boldsymbol{\Delta y}_{t+\tau} - \boldsymbol{\Delta y}_{t-\tau})}{2\sum_{\tau=1}^{D} \tau^2}$$

Normalised log-energy, along with its delta and delta-delta parameters, is commonly added to give the final feature vector. Energy is normalised as a simple gain control. Thus

$$\boldsymbol{o}_t = \begin{bmatrix} \boldsymbol{y}_t \\ e_t \\ \boldsymbol{\Delta y}_t \\ \Delta e_t \\ \boldsymbol{\Delta^2 y}_t \\ \Delta^2 e_t \end{bmatrix}$$

is a commonly used feature vector. For 12 MFCCs, this leads to a **39-dimensional** feature vector.

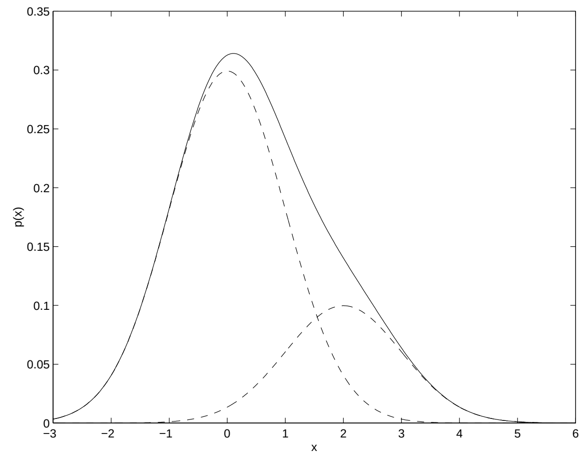
Mixture Output Distributions

Using a single (multivariate) Gaussian to model the output distribution may be poor and may be better 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 weight, or prior. For this to be a pdf it is necessary that

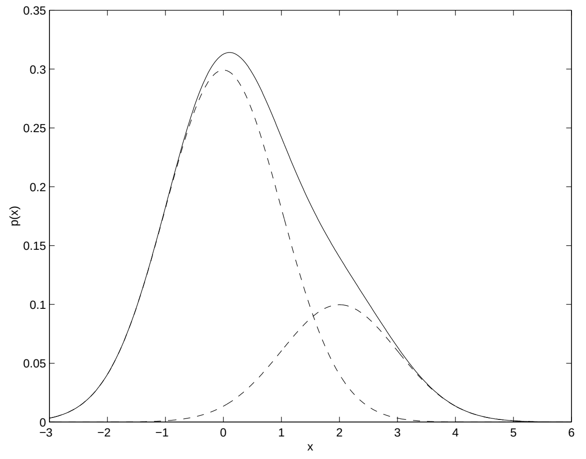
$$\sum_{m=1}^{M} c_{jm} = 1 \quad \text{and} \quad c_{jm} \geq 0$$

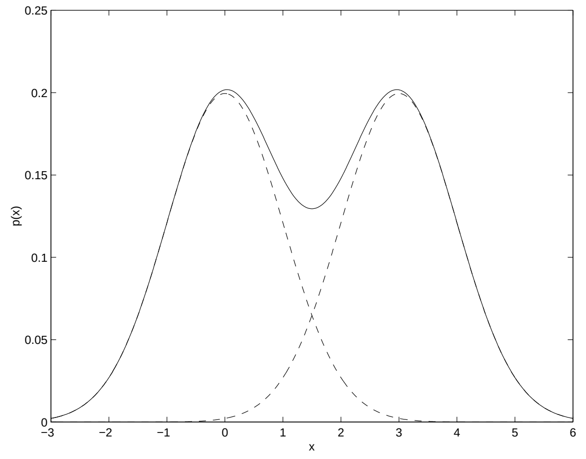


A two component Gaussian mixture distribution is shown. Using Gaussian mixtures it is possible to approximate any distribution (provided you have enough components). Mixture distributions allow very flexible modelling of the acoustic vectors associated with a state.

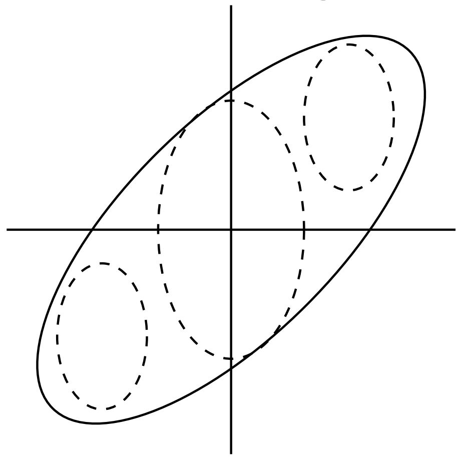
Modelling PDFs

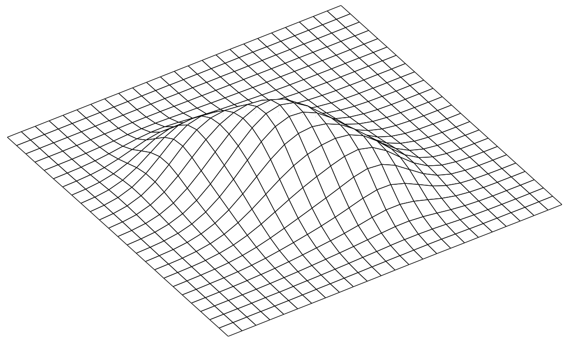
- **Asymmetric** and **Bimodal** distributions





- **Correlation** Modelling





Model Parameters for HMMs with GMM Output PDFs

The parameters that need to be stored are

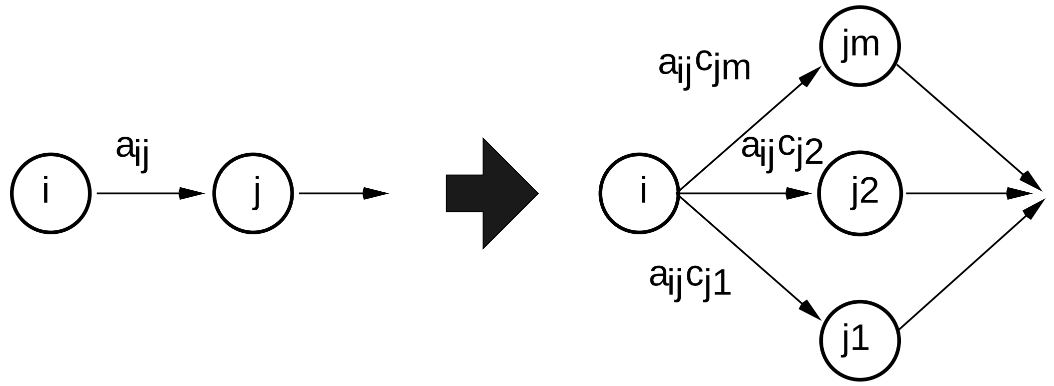
1. the transition matrix (standard HMM);
2. for *each* state the set of weights, means and variances

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

Problem with the use of 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

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

The estimates of the mean and variance will be the similar to the single Gaussian case

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

In addition, it is necessary to estimate the mixture weights. In a similar way to the transition probabilities

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

In terms of the posterior probability of state occupation this becomes

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

Deep Neural Network Acoustic Models

- Deep Neural Networks (DNNs) are Multi-Layer Perceptrons (see 4F10) with many hidden layers (Sigmoid or ReLU units)

- Standard DNNs

  - Model posterior probability of HMM states (1-of-k encoding, softmax)
  - Input is a set of frames (e.g. current frame ± 5)
  - Alignment from GMM-HMM
  - Frame based criterion optimises the cross-entropy criterion
  - Stochastic gradient descent (SGD) via error back propagation (EBP)
  - Initialised using RBM pre-training or EBP (discriminative pre-training)

- DNN-HMM Hybrid models use the probabilities directly divide by state prior to get "scaled likelihoods"

- Tandem models use the DNN to produce features. Use outputs from intermediate hidden layer with few nodes "bottleneck" layer (possibly combined with e.g. PLP). Model by a GMM-HMM as usual.

Summary

- Described Viterbi training of continuous density HMMs (from most-likely state-sequence).
- Described Baum-Welch re-estimation of continuous density HMMs and efficient recursions for generating the forward and backward probabilities.
- The Viterbi algorithm can be used to find the optimal path through an HMM.
- Described the addition of dynamic parameters to the feature vector and how they are calculated.
- Extended the output distribution modelling from single Gaussian distributions to Gaussian mixtures. The extensions to the Baum-Welch re-estimation formulae were also given.
- Deep Neural Network (DNN) models also used to compute HMM output probabilities or generate features for GMM-HMMs.
- HTK toolkit `http://htk.eng.cam.ac.uk` locally developed widely used toolkit for HMM-based speech recognition (includes HTK Book)