MPhil in Machine Learning and Machine Intelligence

Module MLMI2: Speech Recognition

L3: HMM Based Speech Recognition—Introduction

Phil Woodland
pcw@eng.cam.ac.uk

Michaelmas 2021

Cambridge University Engineering Department

Outline

Over the next four lectures we will cover:

- Introduction to Hidden Markov Models (HMMs) for Speech Recognition
- HMM Structure and Assumptions
- Differential Features
- Viterbi Algorithm: the best state sequence
- Basic HMM Search for continuous speech using the Viterbi algorithm
- Outline of Viterbi algorithm implementation & pruning
- Maximum Likelihood HMM Viterbi Training
- Forward-Backward Algorithm
- Baum-Welch Re-estimation
- HMMs with Gaussian Mixture Models (GMM-HMMs)
- Acoustic Models for Large Vocabulary Speech Recognition
- Context-Dependent Acoustic Models
- Introduction to MLMI2 practical

At the end of these lectures, we will have covered enough material for the first part of the speech recognition practical.

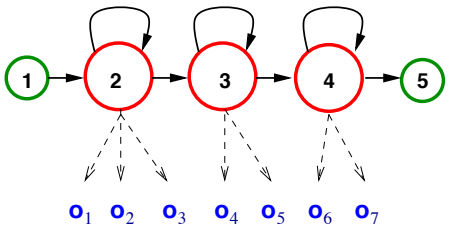
Hidden Markov Models

The HMM is a **generative** model of speech.
In speech recognition is used to find the likelihood that a particular model generated the observed data (i.e. the sequence of speech frames that have been observed).

An HMM is a probabilistic 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, but instead include an initial state distribution).
3. any pair of states i and j can be connected by a transition with probability a_{ij}
4. changes from current state i to state j with probability a_{ij} every frame e.g. each 10 ms.
5. when an emitting state j is entered, acoustic feature vector $\mathbf{o}$ (e.g. of MFCCs) is generated with probability (density) $b_j(\mathbf{o})$
6. the state that the model occupies at any time is a **latent variable**: it is **hidden**!

A particular HMM, $\mathcal{M}$, has parameters $\lambda = \{N, \{a_{ij}\}, \{b_j(\cdot)\}\}$



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 (hidden) state sequence

Note: here using notation from HTK Book.

HMM Assumptions & Likelihood Calculation

Key Assumptions:

1. Probability of moving from one state to another is only dependent on current state. First-order **Markov assumption**. Between state transition probabilities are constant and independent of observations & previously visited states. Not a good model for speech.
2. Probability of observations independent given the state that generated it. **State Conditional Independence assumption**. Previous and following observations do not affect the likelihood. Not true for speech, since speech has a high degree of continuity.
3. The features (**observations**) accurately represent the signal. (Features selected, stationary over 25ms).

Despite its limitations HMMs are the very widely used and successful acoustic models.

For ASR, normally HMMs are **left-to-right** (as in figure: model speech evolution over time).

The joint probability (observations & state-sequence) using state-conditional independence and Markov assumptions 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

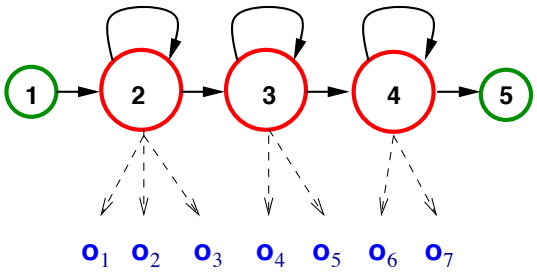
- $x(0)$ is always the entry state 1
- $x(T+1)$ is always the exit state N.

Example Likelihood Calculation

Using the example HMM (right)

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

Assumed a state sequence and hence know 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 (to eliminate dependence on the latent variable: the state at each time)

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

- Evaluating this expression directly is impractical: far too many state sequences.
- However, due to the independence assumptions, **recursive** algorithms exist (**forward-backward algorithm**) that allow $p(\mathbf{O}|\lambda)$ to be computed efficiently while summing over all state sequences.
- Can also calculate $p(\mathbf{O}|\lambda)$ along just **most likely state sequence** (found using the **Viterbi algorithm**).

Output Distributions

The HMM output distribution should be chosen to be able to:

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

A 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:

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

Gaussian distributions with diagonal covariance matrices is 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
- MFCCs allow the use of **diagonal covariance matrices** (far fewer parameters) as a reasonable choice.

Initially we will develop models with multivariate Gaussian distributions as the output probability density. Later we will move on to

- Using Gaussian Mixture Models (**GMM-HMMs**)
- Using pseudo-likelihoods derived from Deep Neural Networks (**DNN-HMMs**).

Note that the HMMs discussed initially for speech recognition are **Continuous Density** HMMs.

- One 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 discrete density HMMs in some examples for convenience/compactness. They can be used for speech if a process of **vector quantisation** is employed (no longer used in speech recognition systems).

Adding Time Differential (Delta) Features

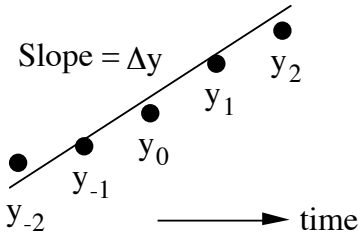
The formulation of the HMM assumes:

- 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 feature vectors used.

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

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



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

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

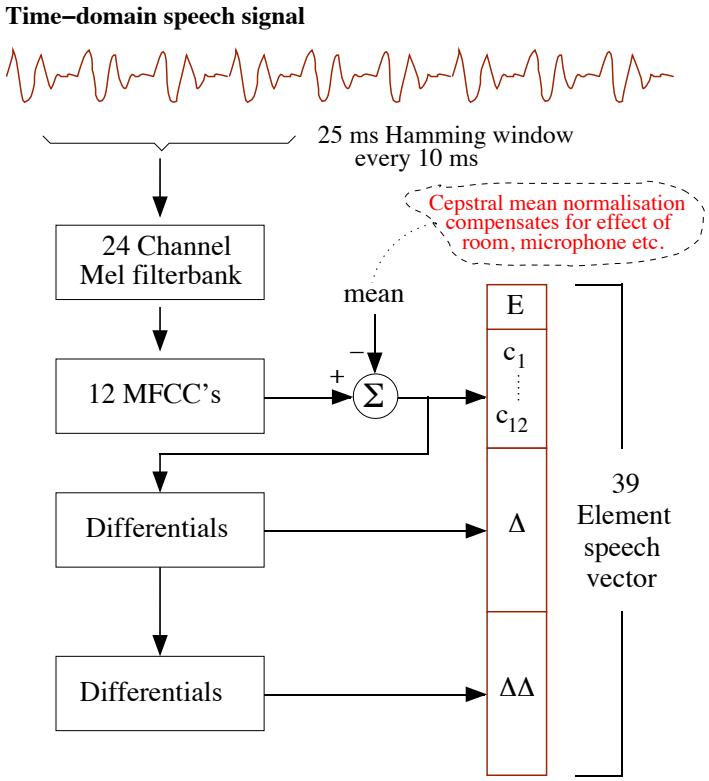
Normalised log-energy, along with its delta and delta-delta parameters, is commonly added to give the final feature vector. Energy is normalised as a simple gain control. Thus

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

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

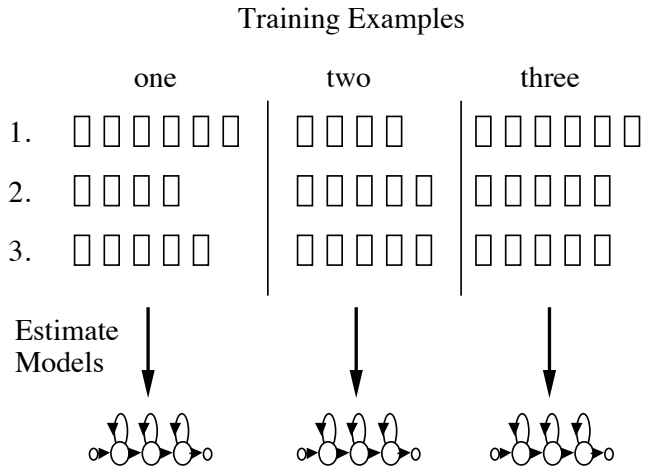
Note that the frame **log energy** (and its differentials) are used in place of c_0 (an alternative). Both need to be **normalised** (e.g. over an utterance) to account for different volume/gain. The cepstra can also be normalised by subtracting their average value (accounts for fixed channel effects) (cepstral mean normalisation).

An ASR front-end



Isolated Word HMM Example: Training

Training for an isolated word speech recogniser is illustrated below.



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. parameters of the HMM are estimated 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, initially 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 an observation sequence $\mathbf{O}$
2. for each HMM word model $\mathcal{M}_i$ compute likelihood of generating $\mathbf{O}$, $p(\mathbf{O}|\mathcal{M}_i)$
3. Assuming equal prior probabilities, the HMM with the highest likelihood identifies the word (otherwise need to account for unequal priors).

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.

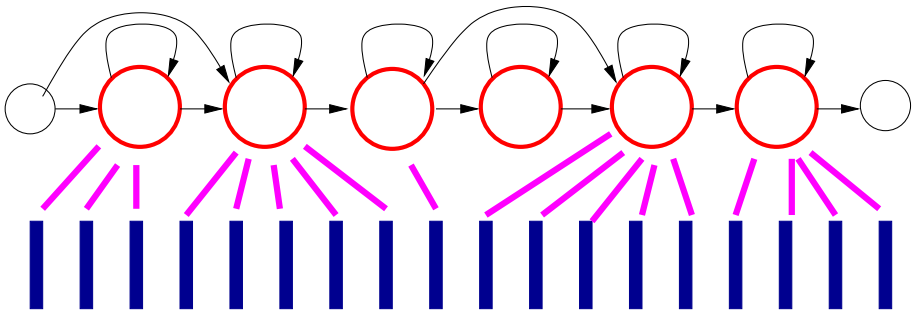
The Best State Sequence

For recognition, the probability of an unknown word is required for each HMM, i.e. $p(\mathbf{O}|\mathcal{M}_i)$ is required for each model $\mathcal{M}_i$. This could be calculated taking into account all possible state sequences, but in practice this is rarely used.

In practice, an estimate of $p(\mathbf{O}|\mathcal{M}_i)$ 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

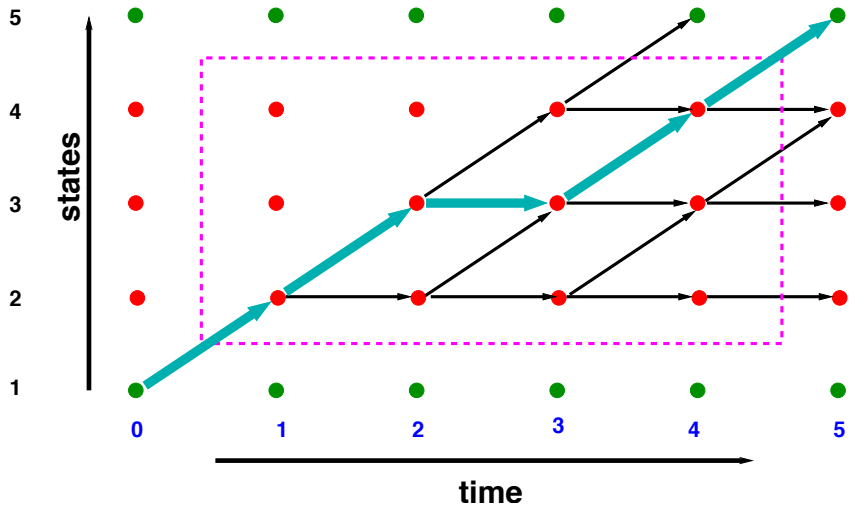


The Trellis

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

Example

Trellis for 5 state HMM (initial/final states non-emitting) and observation sequence 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 model with parameters λ and an observation sequence $\mathbf{O}$ of length T.

It computes

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

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

Define

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

where $X^{(t-1)}$ is the set of all partial paths of length $t-1$. $\phi_j(t)$ represents probability of "best" partial path of length t through the trellis ending in state j.

The partial path probability $\phi_j(t)$ can be calculated using a recursion:

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

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 find most-likely state sequence, necessary to store the local decisions made at each point in Viterbi trellis and then **traceback** along the most likely path at the end of the utterance.

The Viterbi algorithm: Steps

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

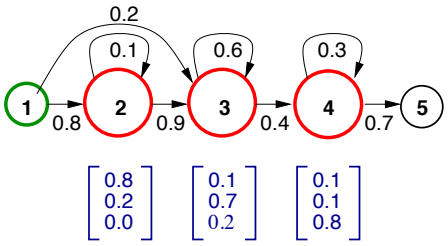
`for` $t = 1, 2, \ldots, T$
... `for` $j = 2, 3, \ldots, N-1$
..... compute $\phi_j(t) = \max_{1 \leq k < N} \left[\phi_k(t-1)a_{kj}\right] b_j(\mathbf{o}_t)$
..... store the predecessor node: $\text{pred}_k(t)$

Termination

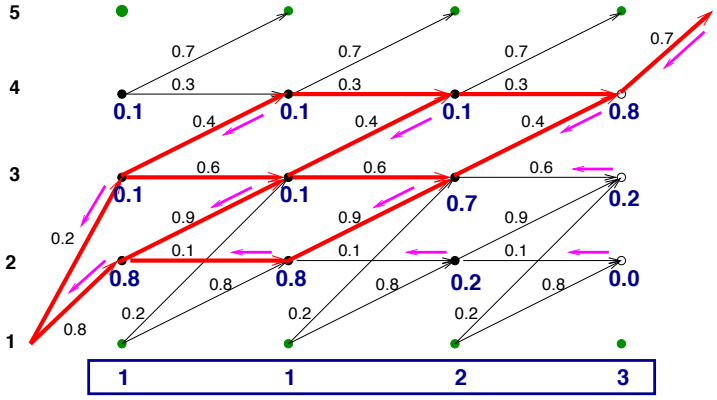
$$p(\mathbf{O}, \mathbf{X}^*|\lambda) = \max_{1<k<N} \phi_k(T)a_{kN}$$

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

Viterbi - Example



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** (even if floats/doubles used!) due to multiplying together a long sequence of numbers each of which are (normally) much less than one.

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

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

Note that the HMM assumptions are key to the efficiency of the Viterbi algorithm.

Maximum Likelihood Estimation for HMMs

In **maximum likelihood** (or ML) parameter estimation, we need to find the HMM model parameters, $\hat{\lambda}$, that **maximise the likelihood** of generating the training data.

Training data is a set of R utterances $\mathbf{O}^{(r)}$ for the speech units (e.g. a word) represented by a model with parameter λ and that each training utterance is of length $T^{(r)}$.
Objective is to find

$$\hat{\lambda} = \arg\max_{\lambda} \left\{ \prod_{r=1}^{R} p(\mathbf{O}^{(r)}|\lambda) \right\}$$

Often assume for simplicity a single training utterance with T frames i.e. $\mathbf{O} = \boldsymbol{o}_1, \ldots, \boldsymbol{o}_T$

- Algorithms are **iterative** because of the hidden nature of an HMM (& if Gaussian mixture models are used for output distributions).
- Two variants will be discussed:
  - First based on best state sequence: **Viterbi** training
  - Then based on all possible state sequences: **Baum-Welch** training
- Each of these operates iteratively & on each iteration performs
  - refine "alignment"
  - given alignment update parameters

Both approaches can be viewed in terms of the Expectation-Maximisation (E-M) algorithm.

Note that GMM-HMM based discriminative training will be discussed later in the course, but even in this case it is necessary to initially perform maximum likelihood training.

HMM-Based ASR Intro Summary

- An HMM is a generative model that assigns a likelihood to a sequence of observation vectors (e.g. MFCCs)
- An HMM represents variability in duration through its state transition structure $\{a_{ij}\}$ and variability in spectra through its output distributions $\{b_j(\cdot)\}$.
- Key HMM assumptions are first order Markov structure, and state-conditional independence.
- Assumptions make efficient computation possible for both recognition and training.
- To make effect of assumptions less severe, typically append first and second order deltas to observation vector.
- For isolated word recognition with equal priors, words can be recognised by finding the word HMM which yields the highest likelihood for the speech data.
- Viterbi algorithm can find the most likely state sequence (path) through an HMM.
- Same overall approach to recognition can be extended to continuous speech recognition
- Parameter estimation based on maximum likelihood estimation: iterative since have to estimate state sequence