LECTURE 22: FUNDAMENTALS OF MARKOV MODELS

- Objectives:
  - Introduce a Markov model
  - Understand the difference between an observable and a hidden Markov model
  - Appreciate the reason we use Markov models: to model temporal evolution of the spectrum (important in speech recognition!)
  - Demonstrate basic calculations

- Demonstrate the infeasibility of these basic calculations for real problems

This material can be found in most speech recognition and pattern recognition textbooks. These notes follow material presented in:

> J. Deller, et. al., *Discrete-Time Processing of Speech Signals*, MacMillan Publishing Co., ISBN: 0-7803-5386-2, 2000.

Another useful reference is:

> L.R. Rabiner and B.W. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, ISBN: 0-13-015157-2, 1993.

The course textbook also follows these traditional references closely.

ECE 8463: FUNDAMENTALS OF SPEECH RECOGNITION

Professor Joseph Picone
Department of Electrical and Computer Engineering
Mississippi State University

email: picone@isip.msstate.edu
phone/fax: 601-325-3149; office: 413 Simrall
URL: http://www.isip.msstate.edu/resources/courses/ece_8463

Modern speech understanding systems merge interdisciplinary technologies from Signal Processing, Pattern Recognition, Natural Language, and Linguistics into a unified statistical framework. These systems, which have applications in a wide range of signal processing problems, represent a revolution in Digital Signal Processing (DSP). Once a field dominated by vector-oriented processors and linear algebra-based mathematics, the current generation of DSP-based systems rely on sophisticated statistical models implemented using a complex software paradigm. Such systems are now capable of understanding continuous speech input for vocabularies of hundreds of thousands of words in operational environments.

In this course, we will explore the core components of modern statistically-based speech recognition systems. We will view speech recognition problem in terms of three tasks: signal modeling, network searching, and language understanding. We will conclude our discussion with an overview of state-of-the-art systems, and a review of available resources to support further research and technology development.

Tar files containing a compilation of all the notes are available. However, these files are large and will require a substantial amount of time to download. A tar file of the html version of the notes is available here. These were generated using wget:

```
wget -np -k -m
http://www.isip.msstate.edu/publications/courses/ece_8463/lectures/current
```

A pdf file containing the entire set of lecture notes is available here. These were generated using Adobe Acrobat.

Questions or comments about the material presented here can be directed to help@isip.msstate.edu.

LECTURE 22: FUNDAMENTALS OF MARKOV MODELS

- Objectives:
  - Introduce a Markov model
  - Understand the difference between an observable and a hidden Markov model
  - Appreciate the reason we use Markov models: to model temporal evolution of the spectrum (important in speech recognition!)

- Demonstrate basic calculations

- Demonstrate the infeasibility of these basic calculations for real problems

This material can be found in most speech recognition and pattern recognition textbooks. These notes follow material presented in:

J. Deller, et. al., *Discrete-Time Processing of Speech Signals*, MacMillan Publishing Co., ISBN: 0-7803-5386-2, 2000.

Another useful reference is:

L.R. Rabiner and B.W. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, ISBN: 0-13-015157-2, 1993.

The course textbook also follows these traditional references closely.

A SIMPLE MARKOV MODEL FOR WEATHER PREDICTION

What is a first-order Markov chain?

$$P[q_t = j | (q_{t-1} = i, q_{t-2} = k, \ldots)] = P[q_t = j | q_{t-1} = i]$$

We consider only those processes for which the right-hand side is independent of time:

$$a_{ij} = P[q_t = j | q_{t-1} = i] \qquad 1 \le i, j \le N$$

with the following properties:

$$a_{ij} \ge 0 \qquad \forall j, i$$

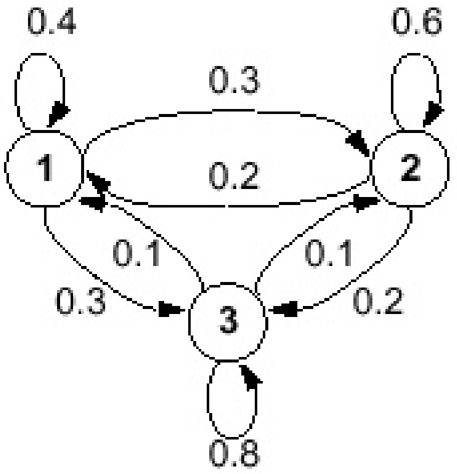
$$\sum_{j=1}^{N} a_{ij} = 1 \qquad \forall i$$

The above process can be considered observable because the output process is a set of states at each instant of time, where each state corresponds to an observable event.

Later, we will relax this constraint, and make the output related to the states by a second random process.

Example: A three-state model of the weather

State 1: precipitation (rain, snow, hail, etc.)
State 2: cloudy
State 3: sunny



BASIC CALCULATIONS

Example: What is the probability that the weather for eight consecutive days is "sun-sun-sun-rain-rain-sun-cloudy-sun"?

Solution:

O = sun	sun	sun	rain	rain	sun	cloudy	sun
3	3	3	1	1	3	2	3

$$\begin{aligned} P(\overline{O} | Model) &= P[3]P[3|3]P[3|3]P[1|3]P[1|1]P[3|1]P[2|3]P[3|2] \\ &= \pi_3 a_{33} a_{31} a_{11} a_{13} a_{32} a_{23} \\ &= 1.536 \times 10^{-4} \end{aligned}$$

Example: Given that the system is in a known state, what is the probability that it stays in that state for *d* days?

O = i i i ... i j

$$\begin{aligned} P(\overline{O} | Model, q_1 = i) &= P(\overline{O}, q_1 = i | Model) / P(q_1 = i) \\ &= \pi_i a_{ii}^{d-1}(1 - a_{ii}) / \pi_i \\ &= a_{ii}^{d-1}(1 - a_{ii}) \\ &= p_i(d) \end{aligned}$$

Note the exponential character of this distribution.

We can compute the expected number of observations in a state given that we started in that state:

$$\bar{d}_i = \sum_{d=1}^{\infty} d p_i(d) = \sum_{d=1}^{\infty} d a_{ii}^{d-1}(1 - a_{ii}) = \frac{1}{1 - a_{ii}}$$

Thus, the expected number of consecutive sunny days is (1/(1-0.8)) = 5; the expected number of cloudy days is 2.5, etc.

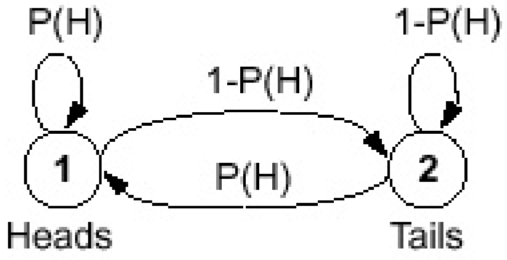
What have we learned from this example?

"HIDDEN" MARKOV MODELS

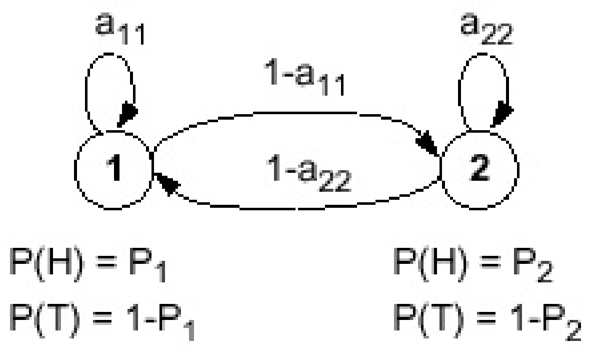
Consider the problem of predicting the outcome of a coin toss experiment. You observe the following sequence:

$$\bar{O} = (HHTTTHTTH \ldots H)$$

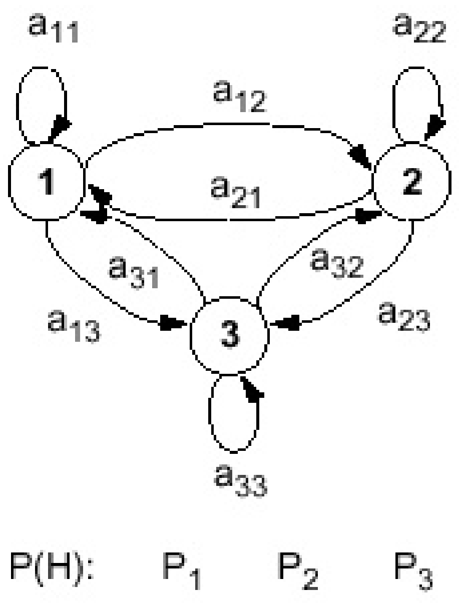
What is a reasonable model of the system?



1-Coin Model
(Observable Markov Model)
O = H H T T H T H H T T H ...
S = 1 1 2 2 1 2 1 1 2 2 1 ...



2-Coins Model
(Hidden Markov Model)
O = H H T T H T H H T T H ...
S = 1 1 2 2 1 2 1 1 2 2 1 ...

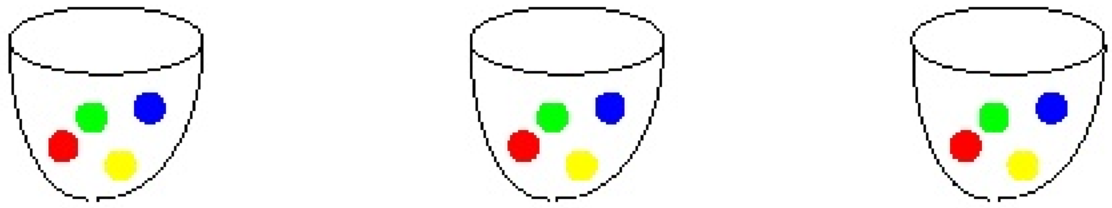


3-Coins Model
(Hidden Markov Model)
O = H H T T H T H H T T H ...
S = 3 1 2 3 3 1 1 2 3 1 3 ...

P(T): $1-P_1$ $1-P_2$ $1-P_3$

DOUBLY STOCHASTIC SYSTEMS

The Urn-and-Ball Model



P(red)	$= b_1(1)$	P(red)	$= b_2(1)$	P(red)	$= b_3(1)$
P(green)	$= b_1(2)$	P(green)	$= b_2(2)$	P(green)	$= b_3(2)$
P(blue)	$= b_1(3)$	P(blue)	$= b_2(3)$	P(blue)	$= b_3(3)$
P(yellow)	$= b_1(4)$	P(yellow)	$= b_2(4)$	P(yellow)	$= b_3(4)$
...		...		...	

$\vec{O}$ = {green, blue, green, yellow, red, ..., blue}

How can we determine the appropriate model for the observation sequence given the system above?

BASIC ELEMENTS OF A HIDDEN MARKOV MODEL

- N — the number of states
- M — the number of distinct observations per state
- The state-transition probability distribution $\underline{A} = \{a_{ij}\}$
- The output probability distribution $\underline{B} = \{b_j(k)\}$
- The initial state distribution $\pi = \{\pi_i\}$

We can write this succinctly as: $\lambda = (\underline{A}, \underline{B}, \pi)$

Note that the probability of being in any state at any time is completely determined by knowing the initial state and the transition probabilities:

$$\pi(t) = \underline{A}^{t-1}\pi$$

Two basic problems:

(1) how do we train the system?

(2) how do we estimate the probability of a given sequence (recognition)?

This gives rise to a third problem:

If the states are hidden, how do we know what states were used to generate a given output?

How do we represent continuous distributions (such as feature vectors)?

MATRIX CALCULATIONS FOR DISCRETE HMMS

The *discrete observation* HMM is restricted to the production of a finite set of discrete observations (or sequences). The output distribution at any state is given by:

$$b(k, i) \equiv P(\underline{y}(t) = k \,|\, \underline{x}(t) = i)$$

The observation probabilities are assumed to be independent of time. We can write the probability of observing a particular observation, $\underline{y}(t)$, as:

$$b(y(t) | i) \equiv P(\underline{y}(t) = y(t) \,|\, \underline{x}(t) = i)$$

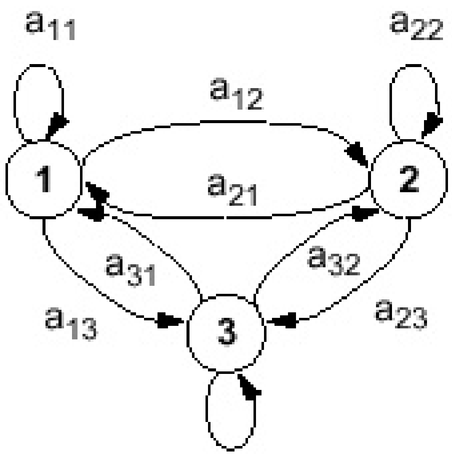
The observation probability distribution can be represented as a matrix whose dimension is K rows x S states.
We can define the observation probability vector as:

$$\boldsymbol{p}(t) = \begin{bmatrix} P(\underline{y}(t) = 1) \\ P(\underline{y}(t) = 2) \\ \dots \\ P(\underline{y}(t) = K) \end{bmatrix}, \quad \text{or,} \quad \boldsymbol{p}(t) = \boldsymbol{B}\pi(t) = \boldsymbol{B}\boldsymbol{A}^{t-1}\pi(1)$$

The mathematical specification of an HMM can be summarized as:

$$\boldsymbol{M} = \{S, \pi(1), \boldsymbol{A}, \boldsymbol{B}, \{\boldsymbol{y}_k, 1 \le k \le K\}\}$$

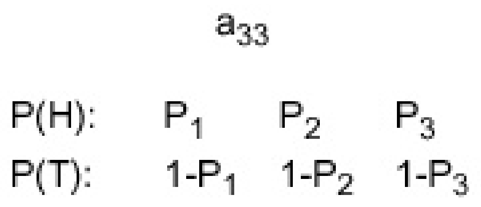
For example, reviewing our coin-toss model:



$$S = 3$$

$$\pi(1) = \begin{Bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{Bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$



P(H):	P_1	P_2	P_3
P(T):	$1-P_1$	$1-P_2$	$1-P_3$

$$\boldsymbol{B} = \begin{bmatrix} P_1 & P_2 & P_3 \\ 1-P_1 & 1-P_2 & 1-P_3 \end{bmatrix}$$

RECOGNITION USING DISCRETE HMMS

Denote any partial sequence of observations in time by:

$$y_{t_1}^{t_2} \equiv \{y(t_1), y(t_1+1), y(t_1+2), \ldots, y(t_2)\}$$

The forward partial sequence of observations at time t is

$$y_1^t \equiv \{y(1), y(2), \ldots, y(t)\}$$

The backward partial sequence of observations at time t is

$$y_{t+1}^T \equiv \{y(t+1), y(t+2), \ldots, y(T)\}$$

A complete set of observations of length T is denoted as $y \equiv y_1^T$.

What is the likelihood of an HMM?

We would like to calculate $P(M|\underline{y} = y)$ — however, we can't. We can (see the introductory notes) calculate $P(\underline{y} = y|M)$. Consider the brute force method of computing this. Let $\vartheta = \{i_1, i_2, \ldots, i_T\}$ denote a specific state sequence. The probability of a given observation sequence being produced by this state sequence is:

$$P(y|\vartheta, M) = b(y(1)|i_1)b(y(2)|i_2)\ldots b(y(T)|i_T)$$

The probability of the state sequence is

$$P(\vartheta|M) = P(\underline{x}(1) = i_1)a(i_2|i_1)a(i_3|i_2)\ldots a(i_T|i_{T-1})$$

Therefore,

$$P(y,(\vartheta|M)) = P(\underline{x}(1) = i_1)a(i_2|i_1)a(i_3|i_2)\ldots a(i_T|i_{T-1})$$
$$x \;\; b(y(1)|i_1)b(y(2)|i_2)\ldots b(y(T)|i_T)$$

To find $P(y|M)$, we must sum over all possible paths:

$$P(y|M) = \sum_{\forall\vartheta} P(y,(\vartheta|M))$$

This requires $O(2TS^T)$ flops. For $S = 5$ and $T = 100$, this gives about 1.6×10^{72} computations per HMM!

1.6×10 computations per HMM!

1.5: HMM Methods in Speech Recognition

Renato De Mori[†] & Fabio Brugnara[‡]

[†] McGill University, Montréal, Québéc, Canada

[‡] Istituto per la Ricerca Scientifica e Tecnologica, Trento, Italy

Modern architectures for Automatic Speech Recognition (ASR) are mostly software architectures generating a sequence of word hypotheses from an acoustic signal. The most popular algorithms implemented in these architectures are based on statistical methods. Other approaches can be found in [WL90] where a collection of papers describes a variety of systems with historical reviews and mathematical foundations.

A vector $\mathbf{y}_t$ of acoustic features is computed every 10 to 30 msec. Details of this component can be found in section . Various possible choices of vectors together with their impact on recognition performance are discussed in [HUGN93].

Sequences of vectors of acoustic parameters are treated as observations of acoustic word models used to compute $p(\mathbf{y}_1^T|W)$, the probability of observing a sequence $\mathbf{y}_1^T$ of vectors when a word sequence **W** is pronounced. Given a sequence $\mathbf{y}_1^T$, a word sequence $\widehat{W}$ is generated by the ASR system with a search process based on the rule:

$$\widehat{W} = \arg\max_{W} \; p(\mathbf{y}_1^T|W)\; p(W)$$

$\widehat{W}$ corresponds to the candidate having maximum a-posteriori probability (MAP). $p(\mathbf{y}_1^T|W)$ is computed by Acoustic Models (AM), while $p(W)$ is computed by Language Models (LM).

For large vocabularies, search is performed in two steps. The first generates a word lattice of the *n-best* word sequences with simple models to compute approximate likelihoods in real-time. In the second step more accurate likelihoods are compared with a limited number of hypotheses. Some systems generate a single word sequence hypothesis with a single step. The search produces an hypothesized word sequence if the task is dictation. If the task is understanding then a conceptual structure is obtained with a process that may involve more than two steps. Ways for automatically learning and extracting these structures are described in [KDMM94].

1.5.1: Acoustic Models

In a statistical framework, an inventory of elementary probabilistic models of basic linguistic units (e.g., phonemes) is used to build word representations. A sequence of acoustic parameters, extracted from a spoken utterance, is seen as a realization of a concatenation of elementary processes described by hidden Markov models (HMMs). An HMM is a composition of two stochastic processes, a *hidden* Markov chain, which accounts for *temporal* variability, and an observable process, which accounts for *spectral* variability. This combination has proven to be powerful enough to cope with the most important sources of speech ambiguity, and flexible enough to allow the realization of recognition systems with dictionaries of tens of thousands of words.

Structure of a Hidden Markov Model

A hidden Markov model is defined as a pair of stochastic processes $(\boldsymbol{X}, \boldsymbol{Y})$. The $\boldsymbol{X}$ process is a first order Markov chain, and is not directly observable, while the $\boldsymbol{Y}$ process is a sequence of random variables taking values in the space of acoustic parameters, or *observations*.

Two formal assumptions characterize HMMs as used in speech recognition. The *first-order Markov hypothesis* states that history has no influence on the chain's future evolution if the present is specified, and the *output independence hypothesis* states that neither chain evolution nor past observations influence the present observation if the last chain transition is specified.

Letting $y \in \mathcal{Y}$ be a variable representing observations and $i, j \in \mathcal{X}$ be variables representing model states, the model can be represented by the following parameters:

$$\begin{array}{lcll} A & \equiv & \{a_{i,j} | i, j \in \mathcal{X}\} & \text{transition probabilities} \\ B & \equiv & \{b_{i,j} | i, j \in \mathcal{X}\} & \text{output distributions} \\ \Pi & \equiv & \{\pi_i | i \in \mathcal{X}\} & \text{initial probabilities} \end{array}$$

with the following definitions:

$$\begin{array}{rcl} a_{i,j} & \equiv & p(X_t = j | X_{t-1} = i) \\ b_{i,j}(y) & \equiv & p(Y_t = y | X_{t-1} = i, X_t = j) \\ \pi_i & \equiv & p(X_0 = i) \end{array}$$

A useful tutorial on the topic can be found in [Rab89].

Types of Hidden Markov Models

HMMs can be classified according to the nature of the elements of the **B** matrix, which are distribution functions.

Distributions are defined on finite spaces in the so called *discrete HMMs*. In this case, observations are vectors of symbols in a finite alphabet of **N** different elements. For each one of the **Q** vector components, a discrete density $\{w(k)|k = 1, \ldots, N\}$ is defined, and the distribution is obtained by multiplying the probabilities of each component. Notice that this definition assumes that the different components are independent. Figure shows an example of a discrete HMM with one-dimensional observations. Distributions are associated with model transitions.

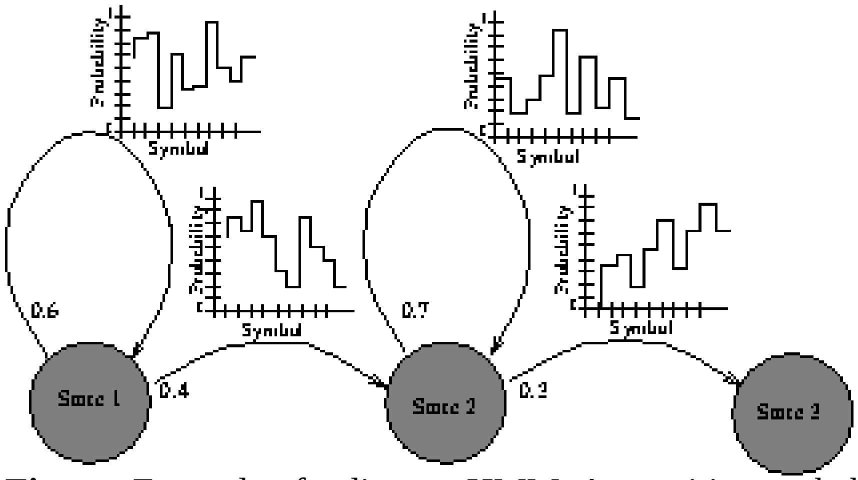


Figure: Example of a discrete HMM. A transition probability and an output distribution on the symbol set is associated with every transition.

Another possibility is to define distributions as probability densities on continuous observation spaces. In this case, strong restrictions have to be imposed on the functional form of the distributions, in order to have a manageable number of statistical parameters to estimate. The most popular approach is to characterize the model transitions with mixtures of base densities **g** of a family **G** having a simple parametric form. The base densities $g \in G$ are usually Gaussian or Laplacian, and can be parameterized by the mean vector and the covariance matrix. HMMs with these kinds of distributions are usually referred to as *continuous HMMs*. In order to model complex distributions in this way a rather large number of base densities has to be used in every mixture. This may require a very large training corpus of data for the estimation of the distribution parameters. Problems arising when the available corpus is not large enough can be alleviated by sharing distributions among transitions of different models. In *semicontinuous HMMs* [HAJ90], for example, all mixtures are expressed in terms of a common set of base densities. Different mixtures are characterized only by different weights.

A common generalization of semicontinuous modeling consists of interpreting the input vector **y** as composed of several components $y[1], \ldots, y[Q]$, each of which is associated with a different set of base distributions. The components are assumed to be statistically independent, hence the distributions associated with model transitions are products of the component density functions.

Computation of probabilities with discrete models is faster than with continuous models, nevertheless it is possible to speed up the mixture densities computation by applying vector quantization (VQ) on the gaussians of the mixtures [Boc93].

Parameters of statistical models are estimated by iterative learning algorithms [Rab89] in which the likelihood of a set of training data is guaranteed to increase at each step.

[BDFK92] propose a method for extracting additional acoustic parameters and performing transformations of all the extracted parameters using a Neural Network (NN) architecture whose weights are obtained by an algorithm that, at the same time, estimates the coefficients of the distributions of the acoustic models. Estimation is driven by an optimization criterion that tries to minimize the overall recognition error.

1.5.2: Word and Unit Models

Words are usually represented by networks of phonemes. Each path in a word network represents a pronunciation of the word.

The same phoneme can have different acoustic distributions of observations if pronounced in different contexts. *Allophone* models of a phoneme are models of that phoneme in different contexts. The decision as to how many allophones should be considered for a given phoneme may depend on many factors, e.g., the availability of enough training data to infer the model parameters.

A conceptually interesting approach is that of *polyphones* [STNE+92]. In principle, an allophone should be considered for every different word in which a phoneme appears. If the vocabulary is large, it is unlikely that there are enough data to train all these allophone models, so models for allophones of phonemes are considered at a different level of detail (word, syllable, triphone, diphone, context independent phoneme). Probability distributions for an allophone having a certain degree of generality can be obtained by mixing the distributions of more detailed allophone models. The loss in specificity is compensated by a more robust estimation of the statistical parameters due to the increasing of the ratio between training data and free parameters to estimate.

Another approach consists of choosing allophones by *clustering* possible contexts. This choice can be made automatically with Classification and Regression Trees (CART). A CART is a binary tree having a phoneme at the root and, associated with each node n_i, a question Q_i about the context. Questions Q_i are of the type, ``Is the previous phoneme a nasal consonant?'' For each possible answer (*YES* or *NO*) there is a link to another node with which other questions are associated. There are algorithms for growing and pruning CARTs based on automatically assigning questions to a node from a manually determined pool of questions. The leaves of the tree may be simply labeled by an allophone symbol. Papers by [BdSG+91] and [HL91] provide examples of the application of this concept and references to the description of a formalism for training and using CARTs.

Each allophone model is an HMM made of states, transitions and probability distributions. In order to improve the estimation of the statistical parameters of these models, some distributions can be the same or tied. For example, the distributions for the central portion of the allophones of a given phoneme can be tied reflecting the fact that they represent the stable (context-independent) physical realization of the central part of the phoneme, uttered with a stationary configuration of the vocal tract.

In general, all the models can be built by sharing distributions taken from a pool of, say, a few thousand cluster distributions called *senones*. Details on this approach can be found in [HH93].

Word models or allophone models can also be built by concatenation of basic structures made by states, transitions and distributions. These units, called *fenones*, were introduced by [BBdS$^+$93b]. Richer models of the same type but using more sophisticated building blocks, called *multones*, are described in [BBdS$^+$93a].

Another approach consists of having clusters of distributions characterized by the same set of Gaussian probability density functions. Allophone distributions are built by considering mixtures with the same components but with different weights [DM94].

1.5.3: Language Models

The probability $p(W)$ of a sequence of words $W = w_1, \ldots, w_L$ is computed by a Language Model (LM). In general $p(W)$ can be expressed as follows:

$$p(W) = p(w_1, .., w_n) = \prod_{i=1}^{n} p(w_i | w_0, .., w_{i-1})$$

Motivations for this approach and methods for computing these probabilities are described in the following section.

H2>1.5.4: Generation of Word Hypotheses

Generation of word hypotheses can result in a single sequence of words, in a collection of the *n-best* word sequences, or in a lattice of partially overlapping word hypotheses.

This generation is a search process in which a sequence of vectors of acoustic features is compared with word models. In this section, some distinctive characteristics of the computations involved in speech recognition algorithms will be described, first focusing on the case of a single-word utterance, and then considering the extension to continuous speech recognition.

In general, the speech signal and its transformations do not exhibit clear indication of word boundaries, so word boundary detection is part of the hypothesization process carried out as a search. In this process, all the word models are compared with a sequence of acoustic features. In the probabilistic framework, ``comparison'' between an acoustic sequence and a model involves the computation of the probability that the model assigns to the given sequence. This is the key ingredient of the recognition process. In this computation, the following quantities are used:

$\alpha_t(y_1^T, i)$:

probability of having observed the partial sequence $\boldsymbol{y}_1^t$ and being in state **i** at time **t**

$$\alpha_t(\boldsymbol{y}_1^T, i) \equiv \begin{cases} p(X_0 = i), & t = 0 \\ p(X_t = i, \boldsymbol{Y}_1^t = \boldsymbol{y}_1^t), & t > 0 \end{cases}$$

$\beta_t(\boldsymbol{y}_1^T, i)$:

probability of observing the partial sequence $\boldsymbol{y}_{t+1}^T$ given that the model is in state **i** at time **t**

$$\beta_t(\boldsymbol{y}_1^T, i) \equiv \begin{cases} p\left(\boldsymbol{Y}_{t+1}^T = \boldsymbol{y}_{t+1}^T | X_t = i\right), & t < T \\ 1, & t = T \end{cases}$$

$\psi_t(\boldsymbol{y}_1^T, i)$:

probability of having observed the partial sequence $\boldsymbol{y}_1^t$ along the best path ending in state **i** at time **t**:

$$\psi_t(\boldsymbol{y}_1^T, i) \equiv \begin{cases} p(X_0 = i), & t = 0 \\ \max\limits_{\boldsymbol{i}_0^{t-1}} p\left(\boldsymbol{X}_0^{t-1} = \boldsymbol{i}_0^{t-1}, X_t = i, \boldsymbol{Y}_1^t = \boldsymbol{y}_1^t\right) & t > 0 \end{cases}$$

α and β can be used to compute the total emission probability $p(\boldsymbol{y}_1^T | W)$ as

$$\begin{aligned} p(\boldsymbol{Y}_1^T = \boldsymbol{y}_1^T) &= \sum_i \alpha_T(\boldsymbol{y}_1^T, i) && (1.1) \\ &= \sum_i \pi_i \beta_0(\boldsymbol{y}_1^T, i) && (1.2) \end{aligned}$$

An approximation for computing this probability consists of following only the path of maximum probability. This can be done with the ψ quantity:

$$\mathbf{Pr}^*[\boldsymbol{Y}_1^T = \boldsymbol{y}_1^T] = \max_i \psi_T(\boldsymbol{y}_1^T, i) \qquad (1.3)$$

The computations of all the above probabilities share a common framework, employing a matrix called a *trellis*, depicted in Figure . For the sake of simplicity, we can assume that the HMM in Figure represents a word and that the input signal corresponds to the pronunciation of an isolated word.

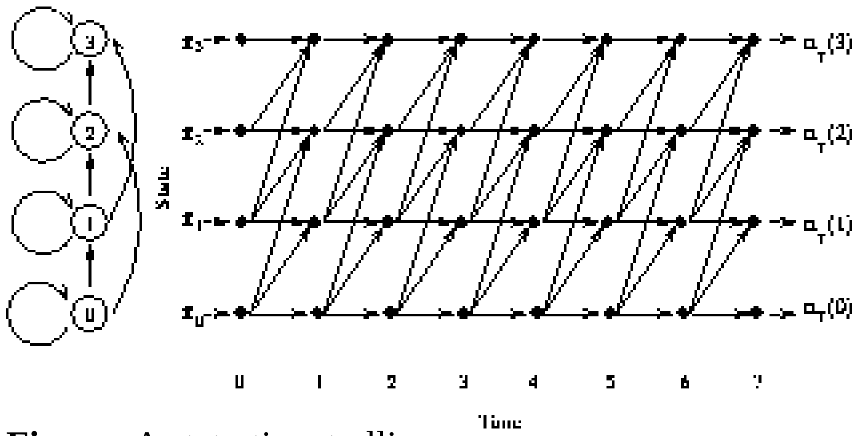


Figure: A state-time trellis.

Every trellis column holds the values of one of the just introduced probabilities for a partial sequence ending at different time instants, and every interval between two columns corresponds to an input frame. The arrows in the trellis represent model transitions composing possible paths in the model from the initial time instant to the final one. The computation proceeds in a column-wise manner, at every time frame updating the scores of the nodes in a column by means of recursion formulas which involve the values of an adjacent column, the transition probabilities of the models, and the values of the output distributions for the corresponding frame. For α and ψ coefficients, the computation starts from the leftmost column, whose values are initialized with the values of π_i, and ends at the opposite side, computing the final value with ( ) or ( ). For the β coefficients, the computation goes from right to left.

The algorithm for computing ψ coefficients is known as the *Viterbi algorithm*, and can be seen as an application of dynamic programming for finding a maximum probability path in a graph with weighted arcs. The recursion formula for its computation is the following:

$$\psi_t(\mathbf{y}_1^T, i) = \begin{cases} \pi_i, & t = 0 \\ \max_j \psi_{t-1}(\mathbf{y}_1^T, j) a_{j,i} b_{j,i}(y_t), & t > 0 \end{cases}$$

By keeping track of the state **j** giving the maximum value in the above recursion formula, it is possible, at the end of the input sequence, to retrieve the states visited by the best path, thus performing a sort of time-alignment of input frames with models states.

All these algorithms have a time complexity $O(MT)$, where **M** is the number of transitions with non-zero probability and **T** is the length of the input sequence. **M** can be at most equal to S^2, where **S** is the number of states in the model, but is usually much lower, since the transition probability matrix is generally sparse. In fact, a common choice in speech recognition is to impose severe constraints on the allowed state sequences, for example $a_{i,j} = 0$ for **j<i, j>i+2**, as is the case of the model in Figure .

In general, recognition is based on a search process which takes into account all the possible segmentations of the input sequence into words, and the a-priori probabilities that the LM assigns to sequences of words.

Good results can be obtained with simple LMs based on bigram or trigram probabilities. As an example, let us consider a bigram language model. This model can be conveniently incorporated into a finite state automaton as shown in Figure , where dashed arcs correspond to transitions between words with probabilities of the LM.

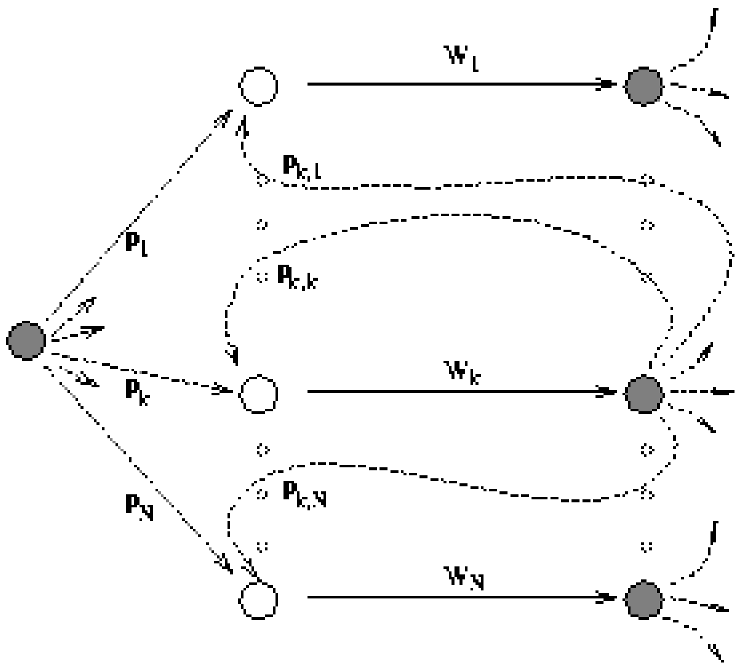


Figure: Bigram LM represented as a weighted word graph. $p_{h,k}$ stands for $p(W_k|W_h)$, p_h stands for $p(W_h)$. The leftmost node is the starting node, rightmost ones are finals.

After substitution of the word-labeled arcs with the corresponding HMMs, the resulting automaton becomes a big HMM itself, on which a Viterbi search for the most probable path, given an observation sequence, can be carried out. The dashed arcs are to be treated as *empty transitions*, i.e., transitions without an associated output distribution. This requires some generalization of the Viterbi algorithm. During the execution of the Viterbi algorithm, a minimum of backtracking information is kept to allow the reconstruction of the best path in terms of word labels. Note that the solution provided by this search is *suboptimal* in the sense that it gives the probability of a single state sequence of the composite model and not the total emission probability of the best word model sequence. In practice, however, it has been observed that the path probabilities computed with the above mentioned algorithms exhibit a dominance property, consisting of a single state sequence accounting for most of the total probability [ME91].

The composite model grows with the vocabulary, and can lead to large search spaces. Nevertheless the uneven distribution of probabilities among different paths can help. It turns out that, when the number of states is large, at every time instant, a large portion of states have an accumulated likelihood which is much less than the highest one, so that it is very unlikely that a path passing through one of these states would become the best path at the end of the utterance. This consideration leads to a complexity reduction technique called *beam search* [NMNP92], consisting of neglecting states whose accumulated score is lower than the best one minus a given threshold. In this way, computation needed to expand *bad* nodes is avoided. It is clear from the naivety of the pruning criterion that this reduction technique has the undesirable property of being *not admissible*, possibly causing the loss of the best path. In practice, good tuning of the beam threshold results in a gain in speed by an order of magnitude, while introducing a negligible amount of search errors.

When the dictionary is of the order of tens of thousands of words, the network becomes too big, and others methods have to be considered.

At present, different techniques exist for dealing with very large vocabularies. Most of them use multi-pass algorithms. Each pass prepares information for the next one, reducing the size of the search space. Details of these methods can be found in [AHH93,ADNS94,MBDW93,KAM+94].

In a first phase a set of candidate interpretations is represented in an object called *word lattice*, whose structure varies in different systems: it may contain only hypotheses on the location of words, or it may carry a record of acoustic scores as well. The construction of the word lattice may involve only the execution of a Viterbi beam-search with memorization of word scoring and localization, as in [ADNS94], or may itself require multiple steps, as in [AHH93,MBDW93,KAM+94]. Since the word lattice is only an intermediate result, to be inspected by other detailed methods, its generation is performed with a bigram language model, and often with simplified acoustic models.

The word hypotheses in the lattice are scored with a more accurate language model, and sometimes with more detailed acoustic models. Lattice rescoring may require new calculations of HMM probabilities [MBDW93], may proceed on the basis of precomputed probabilities only [ADNS94,AHH93], or even exploit acoustic models which are not HMMs [KAM+94]. In [AHH93], the last step is based on an A^* search [Nil71] on the word lattice, allowing the application of a *long distance language model*, i.e., a model where the probability of a word may not only depend on its immediate predecessor. In [ADNS94] a dynamic programming algorithm, using trigram probabilities, is performed.

A method which does not make use of the word lattice is presented in [Pau94]. Inspired by one of the first methods proposed for continuous speech recognition (CSR) [Jel69], it combines both powerful language modeling and detailed acoustic modeling in a single step, performing an A^* based search.

1.5.5: Future Directions

Interesting software architectures for ASR have been recently developed. They provide acceptable recognition performance almost in real time for dictation of large vocabularies (more than 10,000 words). Pure software solutions require, at the moment, a considerable amount of central memory. Special boards make it possible to run interesting applications on PCs.

There are aspects of the best current systems that still need improvement. The best systems do not perform equally well with different speakers and different speaking environments. Two important aspects, namely recognition in noise and speaker adaptation, are discussed in section . They have difficulty in handling out of vocabulary words, hesitations, false starts and other phenomena typical of spontaneous speech. Rudimentary understanding capabilities are available for speech understanding in limited domains. Key research challenges for the future are acoustic robustness, use of better acoustic features and models, use of multiple word pronunciations and efficient constraints for the access of a very large lexicon, sophisticated and multiple language models capable of representing various types of contexts, rich methods for extracting conceptual representations from word hypotheses and automatic learning methods for extracting various types of knowledge from corpora.

Next Up Previous Contents

Next: 1.6: Language Representation **Up:** Spoken Language Input **Previous:** 1.4

htk3

Home | Register | Mailing Lists | Documentation

Home

Getting HTK
- Register
- Change Password
- Download
- ViewCVS

Documentation
- HTKBook
- FAQ
- History of HTK
- CUED LVR Systems
- License

Mailing Lists
- Subscribe
- Accounts
- Archives

Development
- ViewCVS
- Get involved
- Future Plans
- Report a Bug
- Beta version

Links
- ASR Toolkits/Software
- ASR Research Sites
- Speech Companies
- Speech Conferences
- Speech Journals
- ASR Evaluations

Search

Sponsors

What is HTK?

The Hidden Markov Model Toolkit (HTK) is a portable toolkit for building and manipulating hidden Markov models. HTK is primarily used for speech recognition research although it has been used for numerous other applications including research into speech synthesis, character recognition and DNA sequencing. HTK is in use at hundreds of sites worldwide.

HTK consists of a set of library modules and tools available in C source form. The tools provide sophisticated facilities for speech analysis, HMM training, testing and results analysis. The software supports HMMs using both continuous density mixture Gaussians and discrete distributions and can be used to build complex HMM systems. The HTK release contains extensive documentation and examples.

HTK was originally developed at the Speech Vision and Robotics Group of the Cambridge University Engineering Department (CUED) where it has been used to build CUED's large vocabulary speech recognition systems (see CUED HTK LVR). In 1993 Entropic Research Laboratory Inc. acquired the rights to sell HTK and the development of HTK was fully transferred to Entropic in 1995 when the Entropic Cambridge Research Laboratory Ltd was established. HTK was sold by Entropic until 1999 when Microsoft bought Entropic. Microsoft has now licensed HTK back to CUED and is providing support so that CUED can redistribute HTK and provide development support via the HTK3 web site. See History of HTK for more details.

While Microsoft retains the copyright to the existing HTK code, everybody is encouraged to make changes to the source code and contribute them for inclusion in HTK3.

Join the HTK Team at CUED

If you are interested in joining the HTK Team and work on software development or algorithm research (either as an RA or PhD student) send email with your CV to Phil Woodland <pcw@eng.cam.ac.uk>

Current releases NEW

HTK version 3.1 is the current stable release. The old 3.0 release, which corresponds to the Entropic 2.2 version, is still available as well.

Getting HTK

HTK is available for free download but you must first agree to this license. You must then register for a username and password for accessing the HTK CVS and FTP source repositories). Registration is free but does require a valid e-mail address; your password for site access will be sent to this address.

HTK News

16 Jan 2002 (ge):

- HTK3.1 is released. It includes many new features (like support for PLP and VTLN) and bug fixes.

17 May 2001 (ge):

- The meeting of HTK users at ICASSP'01 was a big success. The slides for Gunnar's talk are available here.

20 Apr 2001 (ge):

- A meeting of HTK users will be held on May 10th in Salt Lake City (details)

18 Apr 2001 (ge):

- First beta version of HTK 3.1 available (details)

27 Sep 2000 (rjw):

- HTK3 web-site launched.

Comments and suggestions to htk-mgr@eng.cam.ac.uk

Software

About our Software

Our vision stems from the fact that research commonly suffers from a creative backlog due to rewriting of common functions, and the time spent in debugging such things as file I/O. The ISIP Foundation Classes (IFCs) and software environment are designed to meet this need, providing everything from complex data structures to an abstract file I/O interface.

Our Prototype System is supported across a wide range of platforms including Sun Solaris, Linux, and Cygwin on Windows computers, as long as the minimum software and hardware requirements are met. The latest version of our Prototype System can be downloaded by following our CVS instructions. Then follow the simple quick start guide and you will be on your way.

Download Our Software

- (02/15/02) Production System (v0.0): A research environment that includes a generalized hierarchical Viterbi search-based decoder. Recommended for serious speech and signal processing researchers.
- (03/12/02) Prototype System (v5.12): A cross-word context-dependent LVCSR system. Recommended for speech technologists and application developers.
- (11/29/00) TIDIGITS Toolkit (v5.7): An easy-to-use toolkit that demonstrates the essential steps in building a state-of-the-art speech recognition system. Recommended for novices.

Visit our software release archive for previous release information.

Brain: The same thing we do every night, Pinky. Try to take over the world!

Software-Related Resources

- Documentation: html-based documentation that includes links to the actual source code.
- Tutorials: step-by-step instructions for building a state-of-the-art LVCSR system.
- Demos: conduct an experiment using our remote job submission facility; explore our Java applets.

Consult our legacy software archive for some of our oldies but goodies.

Hidden Markov Model (HMM) Toolbox

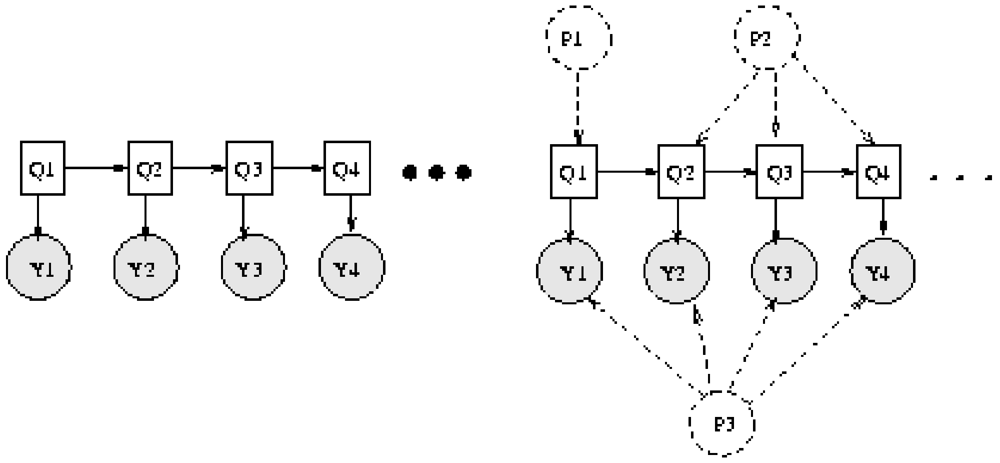
Written by Kevin Murphy, 1998.
Last updated: 14 May 2001.

This toolbox supports inference and learning for HMMs with discrete outputs (dhmm's), Gaussian outputs (ghmm's), or mixtures of Gaussians output (mhmm's). The Gaussians can be full, diagonal, or spherical (isotropic). It also supports discrete inputs, as in a POMDP. The inference routines support filtering, smoothing, and fixed-lag smoothing.

What is an HMM?

An HMM is a Markov chain, where each state generates an observation. You only see the observations, and the goal is to infer the hidden state sequence. HMMs are very useful for time-series modelling, since the discrete state-space can be used to approximate many non-linear, non-Gaussian systems.

HMMs and some common variants (e.g., input-output HMMs) can be concisely explained using the language of Bayesian networks, as we now demonstrate.



Consider the Bayesian network in Figure (a), which represents a hidden Markov model (HMM). (Circles denote continuous-valued random variables, squares denote discrete-valued, clear means hidden, shaded means observed.) This encodes the joint distribution

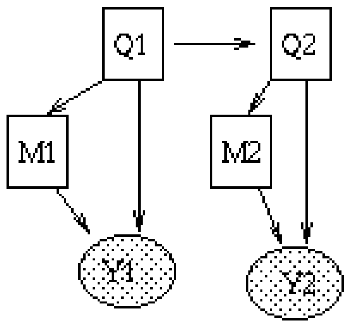
```
P(Q,Y) = P(Q_1) P(Y_1|Q_1) P(Q_2|Q_1) P(Y_2|Q_2) ...
```

For a sequence of length T, we simply ``unroll'' the model for T time steps. In general, such a dynamic Bayesian network (DBN) can be specified by just drawing two time slices (this is sometimes called a 2TBN) --- the structure (and parameters) are assumed to repeat.

The Markov property states that the future is independent of the past given the present, i.e., Q_{t+1} \indep Q_{t-1} | Q_t. We can parameterize this Markov chain using a transition matrix, M_{ij} = P(Q_{t+1}=j | Q_t=i), and a prior distribution, \pi_i = P(Q_1 = i).

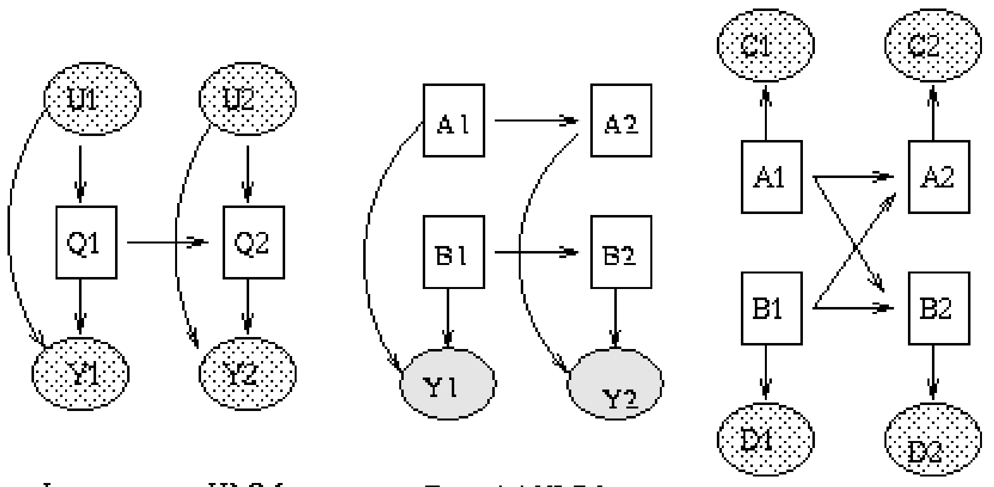
We have assumed that this is a homogeneous Markov chain, i.e., the parameters do not vary with time. This assumption can be made explicit by representing the parameters as nodes: see Figure(b): P1 represents \pi, P2 represents the transition matrix, and P3 represents the parameters for the observation model. If we think of these parameters as random variables (as in the Bayesian approach), parameter estimation becomes equivalent to inference. If we think of the parameters as fixed, but unknown, quantities, parameter estimation requires a separate learning procedure (usually EM). In the latter case, we typically do not represent the parameters in the graph; shared parameters (as in this example) are implemented by specifying that the corresponding CPDs are ``tied".

An HMM is a *hidden* Markov model because we don't see the states of the Markov chain, Q_t, but just a function of them, namely Y_t. For example, if Y_t is a vector, we might define P(Y_t=y|Q_t=i) = N(y; \mu_i, \Sigma_i). A richer model, widely used in speech recognition, is to model the output (conditioned on the hidden state) as a mixture of Gaussians. This is shown below.



HMM with mixture of Gaussians output

Some popular variations on the basic HMM theme are illustrated below (from left to right: an input-output HMM, a factorial HMM, a coupled HMM). (In the input-output model, the CPD P(Q|U) could be a softmax function, or a neural network.) If we have software to handle inference and learning in general Bayesian networks (such as my Bayes Net Toolbox), all of these models becomes trivial to implement.



Input-output HMM Factorial HMM Coupled HMM

This software is designed for simple HMMs. If you want to use these more complicated alternatives, use my Bayes Net Toolbox.

How to use the HMM toolbox

HMMs with discrete outputs

Maximum likelihood parameter estimation using EM (Baum Welch)

The script learn_dhmm_demo.m gives an example of how to learn an HMM with discrete outputs. Let there be Q=2 states and O=2 output symbols. We create random stochastic matrices as follows.

```
O = 3;
Q = 2;
prior0 = normalise(rand(Q,1));
transmat0 = mk_stochastic(rand(Q,Q));
obsmat0 = mk_stochastic(rand(Q,O));
```

Now we sample nex=10 sequences of length T=10 each from this model, to use as training data.

```
data = sample_dhmm(prior0, transmat0, obsmat0, T, nex);
```

Now we make a random guess as to what the parameters are,

```
prior1 = normalise(rand(Q,1));
transmat1 = mk_stochastic(rand(Q,Q));
obsmat1 = mk_stochastic(rand(Q,O));
```

and improve our guess using 5 iterations of EM...

```
max_iter = 5;
[LL, prior2, transmat2, obsmat2] = learn_dhmm(data, prior1, transmat1, obsmat1,
max_iter);
```

LL(t) is the log-likelihood after iteration t, so we can plot the learning curve.

Sequence classification

To evaluate the log-likelihood of a trained model given test data, proceed as follows:

```
loglik = log_lik_dhmm(data, prior, transmat, obsmat)
```

Note: the discrete alphabet is assumed to be {1, 2, ..., O}, where O = size(obsmat, 2). Hence data cannot contain any 0s.

To classify a sequence into one of k classes, train up k HMMs, one per class, and then compute the log-likelihood that each model gives to the test sequence; if the i'th model is the most likely, then declare the class of the sequence to be class i.

Computing the most probable sequence (Viterbi)

First you need to evaluate B(t,i) = P(y_t | Q_t=i) for all t,i:

```
B = mk_dhmm_obs_lik(data, obsmat)
```

Then you can use

```
[path, loglik] = viterbi_path(prior, transmat, B)
```

HMMs with mixture of Gaussians outputs

Maximum likelihood parameter estimation using EM (Baum Welch)

Let us generate nex=10 vector-valued sequences of length T=5; each vector has size O=2.

```
O = 2;
T = 5;
nex = 10;
data = randn(O,T,nex);
```

Now let use fit a mixture of M=2 Gaussians for each of the Q=2 states using K-means.

```
M = 2;
Q = 2;
left_right = 0;
[prior0, transmat0, mixmat0, mu0, Sigma0] =  init_mhmm(data, Q, M, 'diag', 
left_right);
```

Finally, let us improve these parameter estimates using EM.

```
max_iter = 5;
[LL, prior1, transmat1, mu1, Sigma1, mixmat1] = ...
    learn_mhmm(data, prior0, transmat0, mu0, Sigma0, mixmat0, max_iter);
```

Since EM only finds a local optimum, good initialisation is crucial. The procedure implemented in init_mhmm is very crude, and is probably not adequate for real applications...

Sequence classification

To classify a sequence (e.g., of speech) into one of k classes (e.g., the digits 0-9), proceed as in the DHMM case above, but use the following procedure to compute likelihood:

```
loglik = log_lik_mhmm(data, prior, transmat, mixmat, mu, Sigma);
```

Computing the most probable sequence (Viterbi)

First you need to evaluate B(t,i) = P(y_t | Q_t=i) for all t,i:

```
B = mk_mhmm_obs_lik(data, mu, Sigma, mixmat);
```

Finally, use

```
[path, loglik] = viterbi_path(prior, transmat, B);
```

HMMs with Gaussian outputs

This is just like the mixture of Gaussians case, except we have M=1, and hence there is no mixing matrix. The learning routine is called as follows:

```
[LL, prior1, transmat1, mu1, Sigma1] = ...
    learn_mhmm(data, prior0, transmat0, mu0, Sigma0,  max_iter);
```

The classification routine is called as follows:

```
loglik = log_lik_ghmm(data, prior, transmat, mu, Sigma);
```

The likelihood routine is called as

```
B = mk_ghmm_obs_lik(data, mu, Sigma);
```

Online EM for discrete HMMs/ POMDPs

For some applications (e.g., reinforcement learning/ adaptive control), it is necessary to learn a model online. The script online_em_demo gives an example of how to do this.

Other matlab packages for HMMs

- Zoubin Ghahramani has code which is very similar to mine (but doesn't handle mhmm's). He also has code for approximate (variational) inference in factorial HMMs.
- Speech processing toolbox
- More speech processing routines

Recommended reading

- "A tutorial on Hidden Markov Models and selected applications in speech recognition", L. Rabiner, 1989, Proc. IEEE 77(2):257--286.
- "Factorial Hidden Markov Models", Z. Ghahramani and M. Jordan, Machine Learning 29:245--273, 1997.
- Markovian Models for Sequential Data, Y. Bengio, Neural Computing Surveys 2, 129--162, 1999.
- "Statistical Methods for Speech Recognition", F. Jelinek, MIT Press 1997.
- Bibliography on HMMs

Hidden Markov Models for Joint Recognition of Speech and Gesture

ECE497yz Final Project
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
E-mail:gberry@uiuc.edu

Abstract:

Human-computer intelligent interaction (HCII) in virtual environments is a rapidly developing field. The main thrust of research is to create a communication device which is more ``natural'' for humans. I propose the use of gesture and speech recognition to create such an interface in a virtual environment. This project's target application is a simple selection and movement of a virtual object by the computer. This paper address a possible solution for the recognition of the gestures and the speech in such a system.

Greg Berry
9/15/1997