

[Return to Main](#)[Objectives](#)**Review:**[Three Fundamental Problems](#) [The Forward Algorithm](#)**Forward Backward:**[Probability Calculations](#)[Transitions](#)[Application of EM](#)[Reestimation Equations](#)[The Forward-Backward Algorithm](#)**On-Line Resources:**[HLTSurvey: HMM](#)[Cassidy: SR](#)[Software: Discrete HMMs](#)

LECTURE 24: HMM TRAINING

● Objectives:

- Pose parameter reestimation as an unsupervised learning problem
- Introduce the Baum-Welch (Forward-Backward) Algorithm
- Apply EM algorithm to reestimate parameters
- Describe Viterbi training

This lecture combines material from the course textbook:

X. Huang, A. Acero, and H.W. Hon, *Spoken Language Processing - A Guide to Theory, Algorithm, and System Development*, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN: 0-13-022616-5,

2001.

and information found in most standard speech textbooks:

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

[Return to Main](#)**Introduction:**01: Organization
([html](#), [pdf](#))**Speech Signals:**02: Production
([html](#), [pdf](#))03: Digital Models
([html](#), [pdf](#))04: Perception
([html](#), [pdf](#))05: Masking
([html](#), [pdf](#))06: Phonetics and Phonology
([html](#), [pdf](#))07: Syntax and Semantics
([html](#), [pdf](#))**Signal Processing:**08: Sampling
([html](#), [pdf](#))09: Resampling
([html](#), [pdf](#))10: Acoustic Transducers
([html](#), [pdf](#))11: Temporal Analysis
([html](#), [pdf](#))12: Frequency Domain Analysis
([html](#), [pdf](#))13: Cepstral Analysis
([html](#), [pdf](#))14: **Exam No. 1**
([html](#), [pdf](#))15: Linear Prediction
([html](#), [pdf](#))

16: LP-Based Representations

ECE 8463: FUNDAMENTALS OF SPEECH RECOGNITION

Professor Joseph Picone
Department of Electrical and Computer Engineering
Mississippi State Universityemail: 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.

([html](#), [pdf](#))

17: Spectral Normalization

([html](#), [pdf](#))

Parameterization:

18: Differentiation

([html](#), [pdf](#))

19: Principal Components

([html](#), [pdf](#))

20: Linear Discriminant Analysis

([html](#), [pdf](#))

Acoustic Modeling:

21: Dynamic Programming

([html](#), [pdf](#))

22: Markov Models

([html](#), [pdf](#))

23: Parameter Estimation

([html](#), [pdf](#))

24: HMM Training

([html](#), [pdf](#))

25: Continuous Mixtures

([html](#), [pdf](#))

26: Practical Issues

([html](#), [pdf](#))

27: Decision Trees

([html](#), [pdf](#))

28: Limitations of HMMs

([html](#), [pdf](#))

Language Modeling:

LECTURE 24: HMM TRAINING

- Objectives:
 - Pose parameter reestimation as an unsupervised learning problem
 - Introduce the Baum-Welch (Forward-Backward) Algorithm
 - Apply EM algorithm to reestimate parameters
 - Describe Viterbi training

This lecture combines material from the course textbook:

X. Huang, A. Acero, and H.W. Hon, *Spoken Language Processing - A Guide to Theory, Algorithm, and System Development*, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN: 0-13-022616-5, 2001.

and information found in most standard speech textbooks:

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

THREE FUNDAMENTAL PROBLEMS

Recall there are three fundamental problems we must solve:

- **The Evaluation Problem:** Given a model and a set of observations, what was the probability that the model produced these observations?
- **The Decoding Problem:** What was the most likely state sequence in the model that produced a given set of observations?
- **The Learning Problem:** Given a model and a set of observations, how can we adjust the model parameters to increase the probability of the observations (data) given the model?

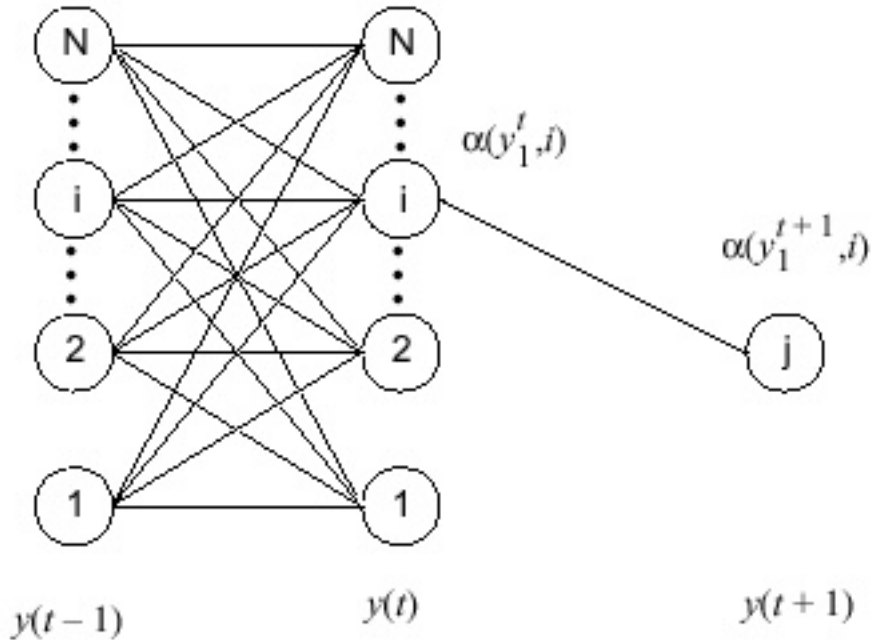
In this lecture, we will focus on the third problem.

THE FORWARD ALGORITHM

The *forward algorithm* begins by defining a “forward-going” partial probability sequence:

$$\alpha(y_1^t) \equiv P(\underline{y}_1^t = y_1^t, \underline{x}(t) = i | \lambda)$$

Let us next consider the contribution to the overall sequence probability made by a single transition:



$$\begin{aligned} \alpha(y_1^{t+1}, j) &= \alpha(y_1^t, i) P(\underline{x}(t+1) = j | \underline{x}(t) = i) \times \\ &\quad P(y(t+1) = y(t+1) | \underline{x}(t+1) = j) \\ &= \alpha(y_1^t, i) a(j|i) b(y(t+1)|j) \end{aligned}$$

Summing over all possibilities for reaching state “j”:

$$\alpha(y_1^{t+1}, j) = \sum_{i=1}^N \alpha(y_1^t, i) a(j|i) b(y(t+1)|j)$$

The recursion is initiated by setting:

$$\alpha(y_1^t, j) = P(\underline{x}(1) = j) b(y(1)|j)$$

We still need to find $P(y|\lambda)$:

$$P(y, \underline{x}(t) = i | \lambda) = \alpha(y_1^t, i) \beta(y_{t+1}^T | i)$$

for any state i . Therefore,

$$P(y|\lambda) = \sum_{i=1}^N \alpha(y_1^T, i)$$

These equations suggest a recursion in which, for each value of t we iterate over ALL states and update $\alpha(y_1^t, j)$. When $t = T$, $P(y|\lambda)$ is computed by summing over ALL states:

The Forward Algorithm

Step 1: Initialization

$$\alpha(y_1^1, j) = \pi_j b(y(1)|j) \quad 1 \leq j \leq N$$

Step 2: Induction

$$\alpha(y_1^{t+1}, j) = \left[\sum_{i=1}^N \alpha(y_1^t, i) a(j|i) \right] b(y(t+1)|j)$$

Step 3: Termination

$$P(y|\lambda) = \sum_{i=1}^N \alpha(y_1^T, i)$$

The complexity of this algorithm is $O(N^2T)$, or for $N = 5$ and $T = 100$, approximately 2500 flops are required (compared to 10^{72} flops for the exhaustive search method).

PROBABILITY CALCULATIONS

Just as the *forward algorithm* computed probabilities recursively:

$$\alpha(y_1^{t+1} | j) = \left[\sum_{i=1}^N \alpha(y_1^t, i) a(j|i) \right] b(y(t+1) | j)$$

we can similarly define a “backward-going” probability calculation:

$$\beta(y_{t+1}^T | i) \equiv P(y_{t+1}^T = y_{t+1}^T | \underline{x}(t) = i, \lambda)$$

This probability can be computed recursively:

$$\beta(y_{t+1}^T | i) = \sum_{j=1}^N \beta(y_{t+2}^T | j) a(j|i) b(y(t+1) | j)$$

This recursion is initialized by:

$$\beta(y_T^T | i) \equiv \begin{cases} 1/N, & \text{if } i \text{ is a legal final state} \\ 0, & \text{otherwise} \end{cases}$$

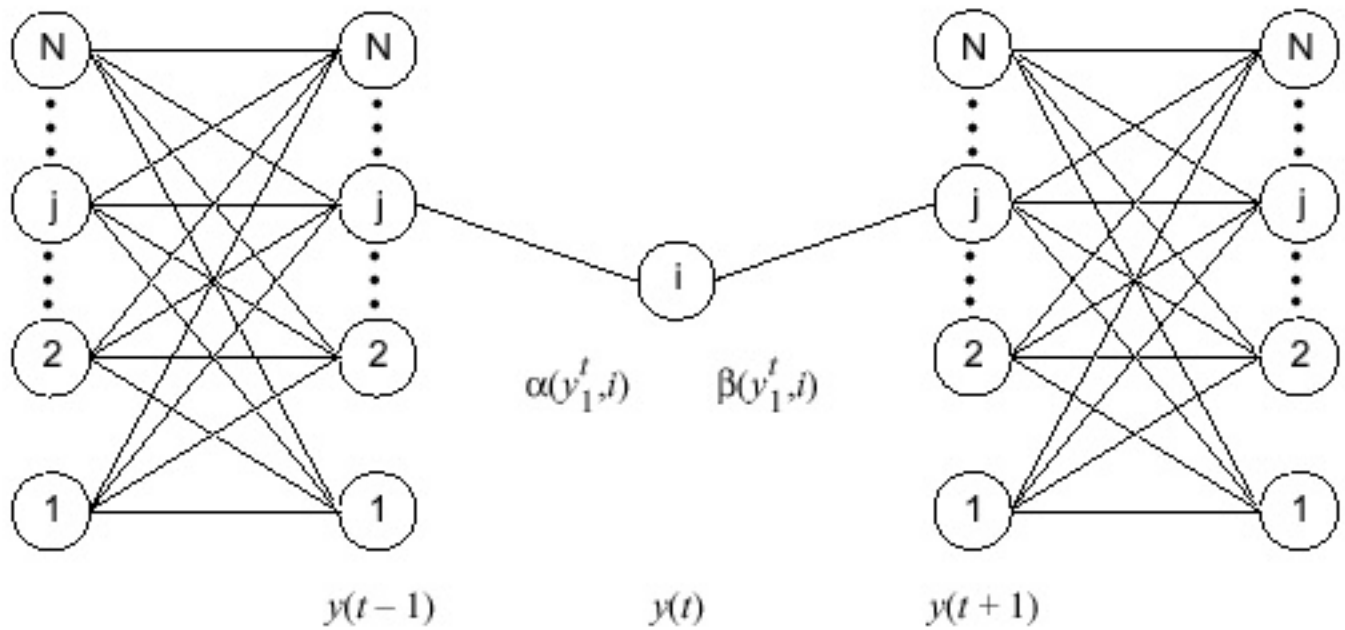
We can now compute $P(y|\lambda)$ in terms of α and β :

$$P(y, \underline{x}(t) = i | \lambda) = \alpha(y_1^t, i) \beta(y_{t+1}^T | i)$$

for any state i . Therefore,

$$P(y|\lambda) = \sum_{i=1}^N \alpha(y_1^t, i) \beta(y_{t+1}^T | i)$$

The calculation can be visualized below:



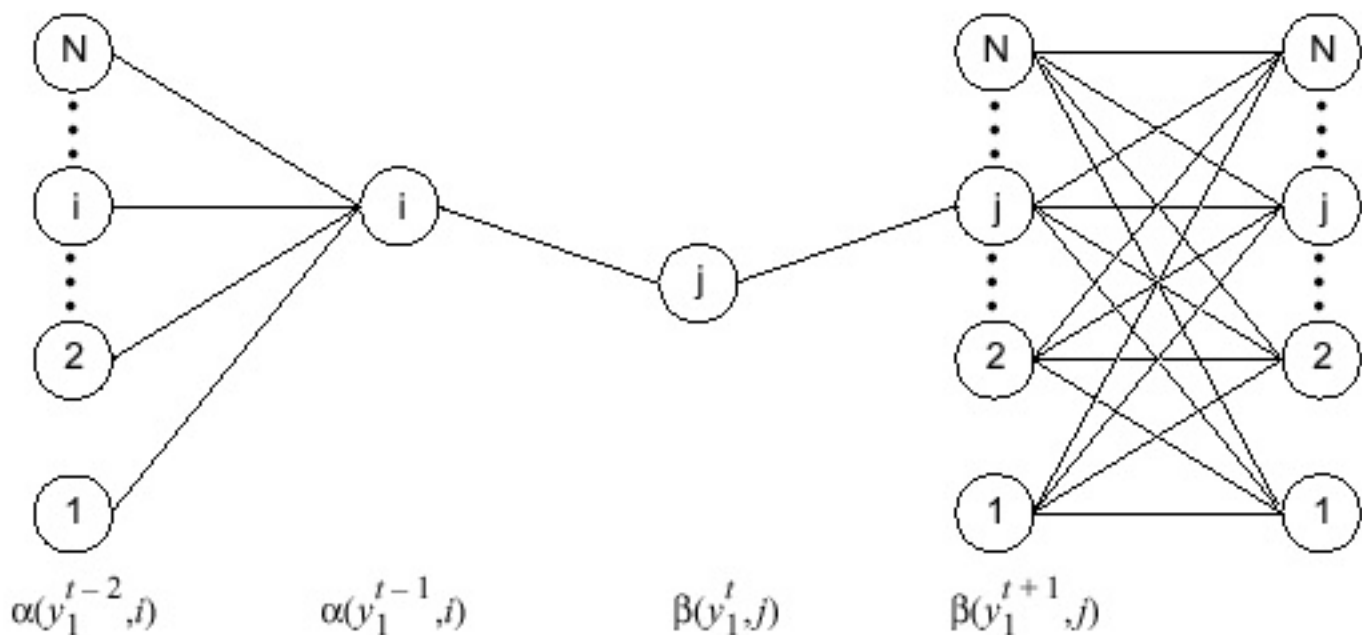
Note that these computations consider all possible state transitions that could have generated the output sequence. This is why this calculation is more expensive than the Viterbi algorithm.

TRANSITION PROBABILITY CALCULATIONS

Define $\gamma_t(j|i)$ as the probability of taking the transition from state i to state j at time t given the model and observation sequence:

$$\begin{aligned} \gamma_t(j|i) &= P(s_{t-1} = i, s_t = j | y_1^T, \lambda) \\ &= \frac{P(s_{t-1} = i, s_t = j, y_1^T | \lambda)}{P(y_1^T | \lambda)} \\ &= \frac{\alpha(y_1^{t-1}, i) a(j|i) b_t(y_t) \beta(y_t^T, j)}{\sum_{k=1}^N \alpha(y_1^T, k)} \end{aligned}$$

This computation is illustrated below:



APPLICATION OF THE EM ALGORITHM

According to the EM algorithm, the maximization process is equivalent to maximizing:

$$Q(\lambda, \bar{\lambda}) = \sum_{i=1}^S \frac{P(y_1^T, S | \lambda)}{P(y_1^T | \lambda)} \log P(y_1^T, S | \bar{\lambda})$$

where:

$$P(y_1^T, S | \lambda) = \prod_{t=1}^T a(s_t | s_{t-1}) b_t(y_t)$$

and

$$\log P(y_1^T, S | \lambda) = \sum_{t=1}^T \log(a(s_t | s_{t-1})) + \sum_{t=1}^T \log(b_t(y_t))$$

This can be rewritten as:

$$Q(\lambda, \bar{\lambda}) = Q_a(\lambda, \hat{a}) + Q_b(\hat{b})$$

where:

$$Q_a(\lambda, \hat{a}) = \sum_{i=1}^N \sum_{j=1}^N \sum_{t=1}^T \frac{P(y_1^T, s_{t-1}=i, s_t=j | \lambda)}{P(y_1^T | \lambda)} \log \hat{a}(j|i)$$

and:

$$Q_b(\lambda, \hat{b}) = \sum_{j=1}^N \sum_{k=1}^M \sum_{t \in (j^t = o^t)} \frac{P(y_1^T, s_t=j | \lambda)}{P(y_1^T | \lambda)} \log \hat{b}_j(k)$$

These terms can be maximized separately under the constraints:

$$\sum_{j=1}^N a(j|i) = 1 \quad \text{and} \quad \sum_{k=1}^M b_j(k) = 1 \quad \forall \text{all } i, j$$

ESTIMATION EQUATIONS

The functions in our auxiliary functions Q are of the form $F(x) = \sum_i y_i \log x_i$,

where $\sum_i x_i = 1$. Using Lagrange multipliers, this function can be shown to

have a maximum value at $x_i = y_i / \sum_i y_i$. The model reestimation equations that result from this optimization are:

$$\hat{a}(j|i) = \frac{\frac{1}{P(y_1^T|\lambda)} \sum_{t=1}^T P(y_1^T, s_{t-1} = i, s_t = j|\lambda)}{\frac{1}{P(y_1^T|\lambda)} \sum_{t=1}^T P(y_1^T, s_{t-1} = i|\lambda)} = \frac{\sum_{t=1}^T \gamma_t(j|i)}{\sum_{t=1}^T \sum_{k=1}^N \gamma_t(k|i)}$$

This is just the ratio of the expected number of transitions from state i to state j and the expected number of transitions from state i .

Similarly,

$$\hat{b}_j(k) = \frac{\frac{1}{P(y_1^T|\lambda)} \sum_{t=1}^T P(y_1^T, s_t = j|\lambda) \delta(y^t, o^t)}{\frac{1}{P(y_1^T|\lambda)} \sum_{t=1}^T P(y_1^T, s_t = j|\lambda)} = \frac{\sum_{t \in (y^t = o^t)} \sum_{i=1}^M \gamma_t(j|i)}{\sum_{t=1}^T \sum_{i=1}^M \gamma_t(j|i)}$$

This is the ratio of the number of times the k^{th} observation vector was emitted from state j and the number of times any observation vector was emitted from state j .

THE FORWARD-BACKWARD ALGORITHM

The Forward Backward Algorithm

Step 1: Initialization: Choose an initial estimate λ .

Step 2: E-step: Compute auxiliary function $Q(\lambda, \hat{\lambda})$ based on λ .

Step 3: M-step: Compute $\bar{\lambda}$ according to the equations for \hat{a} and \hat{b} to maximize the auxiliary Q -function.

Step 4: Iteration: Set $\lambda = \bar{\lambda}$ and repeat steps 2 and 3 until convergence.

The previous derivation assumed one data sequence. To train an HMM from M data sequences, we simply maximize the joint likelihood:

$$\prod_{i=1}^M P(Y_i | \lambda)$$

We can define a partial update for the r^{th} data sequence:

$$\hat{a}(j|i) = \frac{\sum_{r=1}^R \sum_{t=1}^{T^r} \gamma_t^r(j|i)}{\sum_{r=1}^R \sum_{t=1}^{T^r} \sum_{k=1}^N \gamma_t^r(k|i)}$$

The latter equation is important since we can use it to implement HMM training in parallel. We simply run (R/N) files on N processors, and accumulate the sums inside the first summation. After all N jobs complete, we can combine the outputs for the final estimates of the new parameters.

[Next](#)[Up](#)[Previous](#)[Contents](#)[Index](#)Next: [1.6: Language Representation](#) Up: [Spoken Language Input](#) Previous: [1.4](#)

1.5: HMM Methods in Speech Recognition

Renato De Mori[†] & Fabio Brugnara[‡][†] McGill University, Montréal, Québec, 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 (\mathbf{X}, \mathbf{Y}) . The \mathbf{X} process is a first order Markov chain, and is not directly observable, while the \mathbf{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 $\mathbf{y} \in \mathcal{Y}$ 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:

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


with the following definitions:

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

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 \mathbf{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 $\{w(k) | k = 1, \dots, N\}$ 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.

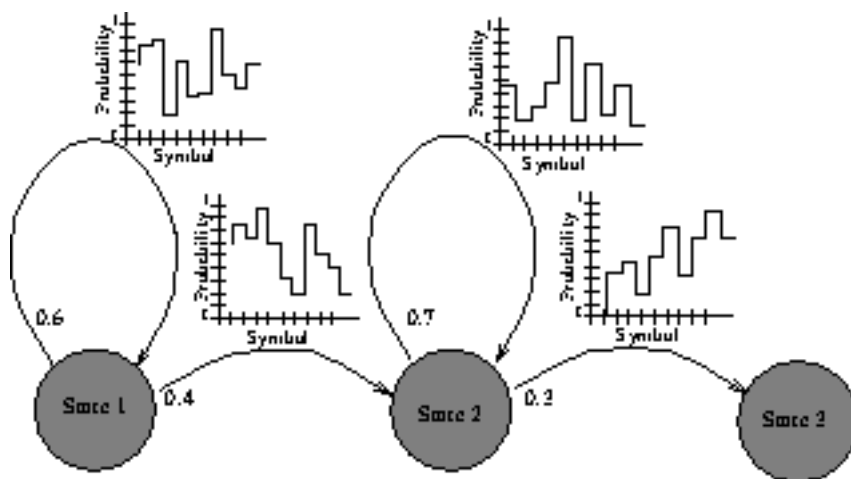


Figure: Example of a discrete HMM. A transition probability and an output distribution on the symbol set is associated with every transition.

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 \mathbf{g} of a family \mathbf{G} having a simple parametric form. The base densities $\mathbf{g} \in \mathbf{G}$ 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 \mathbf{y} as composed of several components $\mathbf{y}[1], \dots, \mathbf{y}[Q]$, 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 $W = w_1, \dots, w_L$ is computed by a Language Model (LM). In general $p(W)$ can be expressed as follows:

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

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:

$$\alpha_t(\mathbf{y}_1^T, i):$$

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

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

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

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

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

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

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

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



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

$$p(\mathbf{Y}_1^T = \mathbf{y}_1^T) = \sum_i \alpha_T(\mathbf{y}_1^T, i) \quad (1.1)$$

$$= \sum_i \pi_i \beta_0(\mathbf{y}_1^T, i) \quad (1.2)$$

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

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

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.

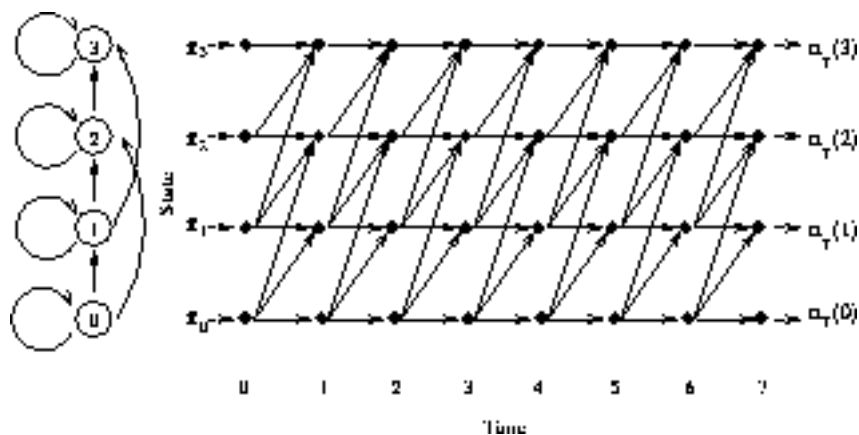


Figure: A state-time trellis.

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 (1) or (2). 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 1.

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.

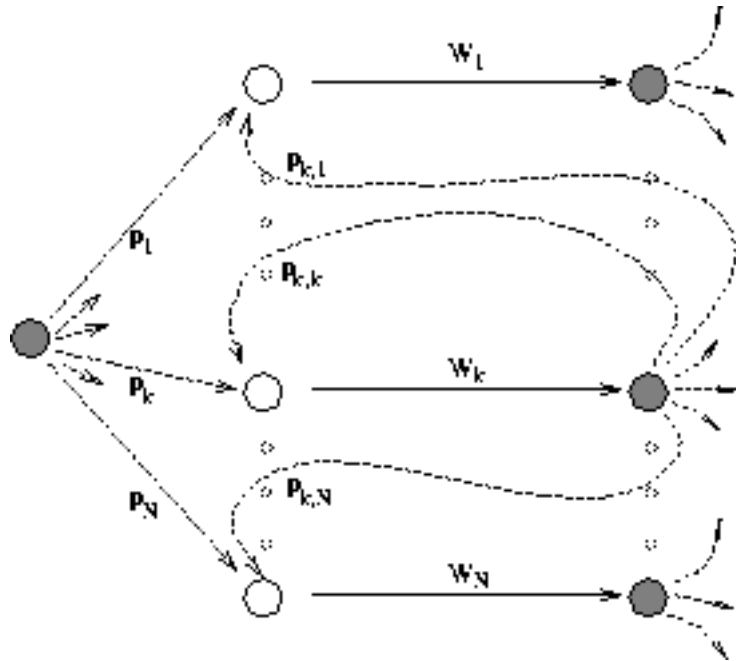


Figure: Bigram LM represented as a weighted word graph. $p_{h,k}$ stands for $p(W_k | W_h)$, p_h stands for $p(W_h)$. The leftmost node is the starting node, rightmost ones are finals.

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 other 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](#)

SLP806: Speech Recognition

Steve Cassidy

Speech Hearing and Language Research Centre, Macquarie University

Sydney

Australia

Steve.Cassidy@mq.edu.au

Copyright © 2001 by Speech Hearing and Language Research Centre

Table of Contents

1. [Outline](#)

[Lecture Outline](#)

2. [Assignments and Assessment](#)

[Assesment](#)

[Gaussian Classification](#)

[Procedure](#)

[Your Report](#)

[Speech/Speaker Recognition](#)

[Basic Methods: DTW](#)

[Data for Analysis](#)

[Vector Quantisation](#)

[Template Matching](#)

[Speaker Recognition](#)

[R Hints](#)

[To Hand In](#)

3. [The Architecture of ASR](#)

[The Problem of Speech Recognition](#)

[Basic Components](#)

[Speech Capture](#)

[Feature Extraction](#)

[Models](#)

[Result Processing](#)

[Speaker Recognition](#)

4. [Feature Extraction for ASR](#)

[Sources of Variability in Speech](#)

[Separating Phoneme Classes](#)

[Linear Predictive Coding](#)

[The Spectrum and the Z-transform](#)

[Convolution as Multiplication](#)

[LPC analysis](#)

[Formants and Smooth Spectra](#)

[Signal Modelling Techniques](#)

[Spectral Shaping](#)

[Spectral Analysis](#)

[Parameter Transforms](#)

5. [Gaussian Classifiers and Distance Measures](#)

[Spaces and Distance Measures](#)

[Gaussian Classifiers](#)

[The Mean Vector](#)

[The Covariance Matrix](#)

[Making use of the Gaussian Model](#)

[Training Statistical Models](#)

[Summary](#)

[Vector Quantisation](#)

[Principal Components Analysis](#)

6. [Matching Patterns in Time](#)

[The Problem with Time](#)

[Dynamic Time Warping](#)

[The DTW Grid](#)

[Optimisations](#)

[The Weighting Function](#)

[Practical DP Matching](#)

[Additional Topics](#)

[Endpoint Detection](#)

[Selecting Templates](#)

[Continuous Speech](#)

7. [Hidden Markov Models](#)

[Temporal Signals](#)

[Applying HMMs](#)

[Building the Models](#)

[Model Parameters](#)

[Models for Continuous Speech](#)

[Training Procedure](#)

8. [Connectionist/HMM Speech Recognition](#)

[Neural Networks](#)

[Temporal Neural Networks](#)

[Hybrid HMM Recognisers](#)

[The Standard Approach Revisited](#)

[The Hybrid HMM/MLP Approach](#)

[Resources](#)

9. [Language Models in ASR](#)

[The Job of a Language Model](#)

[Grammar Based Language Models](#)

[Probabilistic Language Models](#)

[LM Driven Search in ASR](#)

[Viterbi Search](#)

[Stack Decoding](#)

[Multi-Pass Search](#)

[Constructing the Search Space](#)

[Summary](#)

10. [Speaker Identification and Verification](#)

[The Problem](#)

[System Outline](#)

[Speech Parameterisation](#)

[Speaker Modelling](#)

[Decision Making](#)

[Resources](#)

11. [Prosody and Speech Recognition](#)

ProsodyKinds of StressDescribing ProsodyUsing Prosody in ASR**List of Figures**

- 2-1. An example clustering of the first two columns of the melcep data for speaker `a'. Note that the data for vq label `8' has been plotted in white, it fits in between the green '3' cluster and the cyan '5' cluster.
- 5-1. The distribution of vowels in the F1/F2 plane. The red and blue points show two unknown vowels.
- 5-2. The conditional probability density for two types on a single parameter. Point *a* would be assigned to type t_1 while point *b* would be assigned to t_2 .
- 5-3. Some formant data split into 16 clusters using the k-means algorithm.
- 5-4. A two dimensional space showing the direction of maximum variance. PCA seeks to rotate the space such that this direction is parallel to the x-axis.
- 5-5. Two plots showing vowel data. On the left the first two principal components, on the right automatically tracked formants. Each plot shows averaged values over the same set of 172 vowels taken from one male and one female speaker.
- 6-1. An example DTW grid
- 7-1. An example lexical tree which links together phoneme models in a network such that alternate paths through the network represent different words in the lexicon. Figure taken from Deshmukh, Ganapathiraju and Picone (IEEE Signal Processing Magazine, vol. 16, no. 5, September 1999).
- 8-1. A simple neural network. The activation of unit A5 is calculated as the weighted sum of the four units which are connected to it.
- 8-2. A three unit neural network and the subdivision of the space required to solve three simple problems.
- 8-3. The solution to the XOR problem: a hidden node between the input and the output.
- 8-4. Individual phone class posterior probabilities as computed by an MLP for the words *one two three*. Time follows the horizontal axis and the duration of the utterance is 1.5 seconds. Taken from Robinson, Cook, Ellis, Fosler-Lussier, Renals and Williams *Connectionist Speech Recognition of Broadcast News*, Speech Communication, 2000.
- 9-1. Simple overview of the stack decoding algorithm from Deshmukh et al.
- 9-2. An example of the N-best list of hypotheses generated for a simple utterance, and the resulting word graph with N equal to 20. Note that most of the paths are almost equally probable, and only minor variants of each other in terms of segmentation. This indicates the severity of the acoustic confusability in spontaneous, conversational speech recognition (from Deshmukh et al).
- 10-1. Example DET Curve taken from Reynolds and Heck
- 11-1. An example pitch contour with ToBI annotation taken from example ToBI database available on

[the Emu ToBI page.](#)

[Next](#)
Outline


[Home](#)
[Software](#)
[Docs](#)
[Tutorials](#)
[Demos](#)
[Databases](#)
[Dictionaries](#)
[Models](#)
[Research](#)
[Support](#)
[Mailing Lists](#)
[What's New](#)
[Search](#)

Discrete HMM Toolkit

This is a discrete HMM toolkit that we hope will be useful in learning about the fundamental properties of HMMs. The current release contains a program run from the command line. Future releases will provide a Java-based web interface, including a GUI that will help visualize various aspects of HMM theory.

The [code](#) closely parallels the theory presented in:

J.R. Deller, Jr., J.G. Proakis, and J.H.L. Hansen,
 Discrete-Time Processing of Speech Signals,
 MacMillan, 1993, ISBN: 0-02-328301-7.

It is written entirely in C++ (using GNU's gcc compiler), and should fairly easily compile on most machines. To build the code, change directories to src, and execute make. You will need a make utility that supports GNU's make extensions (we recommend GNU's make).

The binary found in the bin directory, hmm.exe, is compiled on a Sun Sparc running Solaris 2.4. The binary supports four major modes of operation:

1. Training

The training mode is typically enabled by first creating a file containing sequences of symbols. For example, to emulate coin toss experiments, you might consider creating a file of random sequences of heads and tails:

```
HHHHHTHTHTHHHT
THTHTHHHTHTHHT
...
```

You can train an HMM on this data by using the following command line:

```
hmm.exe -train -K 2 -S 2 -P 2 -viterbi file.data file.model
```

The arguments are described in the help message (`hmm.exe -help`). The above line generates a 2-state HMM with a 2-symbol codebook, using two passes of training based on the Viterbi algorithm (Baum-Welch training is also available).

The input data is contained in file.data; the output model will be found in file.model.

2. Generation

Once you have a model, you can generate data from the model using the command line:

```
hmm.exe -generate -L 100 file.data file.model
```

Here, the model in file.model is loaded, and 100 random sequences are generated from the model. These can be used to train a new model, or evaluate an existing model.

3. Testing

Given a model, you can evaluate the data set:

```
hmm.exe -test -viterbi file.data file.model
```

This will compute the probability of the data given the model. This allows you to compare various models on the same data, or investigate the effects of various training and testing algorithms.

4. Update

Given a model and a data set, you can update a model:

```
hmm.exe -update -viterbi -P 10 file.data file.model
```

In this mode, the model contained in file.model is loaded, 10 passes of reestimation of the parameters are performed, and the model in file.model is replaced (beware, this overwrites the file).

We hope this is the beginning of some fairly easy to use tools to understand and apply HMMs. This code was written primarily to support simple experiments typically found in introductory chapters on HMMs (coin tosses, selecting colored balls from urns, etc.). It is not intended to be a full-blown speech recognizer - but then again, hopefully, that will be coming soon.

Comments, feedback, bug reports are encouraged. Please send them to help@isip.msstate.edu.

