# Speech and Audio Processing
# Nicholas Evans

---

## Lecture 1: Introduction

Sound attributes:

- Physical quantity: intensity, fundamental frequency, spectral shape, onset/offset, phase
- Perceptual quality: loudness, pitch, timbre, timing, location

Phoneme: minimal unit of speech sound in a language (e.g. iy in feel and ih in fill)

The **S**hort-**T**ime **F**ourier **T**ransform is a transform used to determine the sinusoidal frequency and phase of local sections of a signal (Wikipedia).

$$\mathrm{STFT}\{\mathrm{x[n]}\}(\mathrm{m},\omega) = \mathrm{X}(\mathrm{m},\omega) = \sum_{n=-\infty}^{+\infty} \mathrm{x_m[n]} e^{-j\omega n} = \sum_{n=-\infty}^{+\infty} \mathrm{w(m-n)x[n]} e^{-j\omega n}$$

$w_m[n]$ is a window function (e.g. rectangular)

Logarithmic frame Energy:

$$\log E = \log|X[k]|^2 = \log\left(X_r^2[k] + X_i^2[k]\right)$$

Linear Predictive Coding (LPC)

Speech is characterized with an all-pole filter $H(z) = \frac{X(z)}{E(z)} = \frac{1}{1-\sum_{k=1}^{p} a_k z^{-k}} = \frac{1}{A(k)}$ where $p$ is the order of the LPC analysis.

Starting with $x[n] = \sum_{k=1}^{p} a_k x[n-k] + e[n]$, LPC aims to predict the next sample from previous samples $\tilde{x}[n] = \sum_{k=1}^{p} a_k x[n-k]$ where $e[n]$ is the prediction error.

Yule-Walker equations: $\sum_{j=1}^{p} a_j \Phi_m[i,j] = \Phi_m[i,0]$ for $i = 1, \dots, p$; where the correlation coefficient is defined as $\Phi_m[i,j] = \sum_n x_m[n-i] x_m[n-j]$.

How to choose $p$? Larger values give lower prediction error (in practice between 10 and 14)

Cepstral Processing

This is another way (which works better in practice) to separate source and filter.

How? Homomorphic transformation to transform convolution into sum and then easily subtract $e$: $\hat{x}[n] = D(x[n]) = D(e[n] * h[n]) = \hat{e}[n] + \hat{h}[n]$

Real cepstrum of signal $c[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} \ln\left|X\left(e^{j\omega}\right)\right| e^{j\omega n} d\omega$

For speech signals $c_a[n] = \frac{1}{N}\sum_{n=0}^{N-1} \ln|X_a[k]|\, e^{\frac{j2\pi nk}{N}}$ where $X_a[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{j2\pi nk}{N}}$

Filter: what was said vs. Pitch: who said it (fundamental frequency of the speech).

## Lecture 2: Towards Modeling, Classification and Recognition

Modeling: a way to describe a collection of data.

Vector Quantization: regrouping closest vectors together, e.g. k-means . We should define a distance measure, for example:

- Euclidean distance $d(v, \mu_i) = \|v - \mu_i\|^2 = (v - \mu_i)^T (v - \mu_i)$
- Mahalanobis distance $d(v, \mu_i) = (v - \mu_i)^T \Sigma^{-1} (v - \mu_i)$ (use the information of alignment with the covariance).

## Lecture 3: Deterministic Methods of Automatic Speech Recognition

Two approaches to speech recognition (it's important to know the difference!):

- Deterministic: dynamic time warping
- Stochastic: Hidden Markov Models

### Dynamic Time Warping

An algorithm to measure similarities between two time series (Video).

Different acoustic performances are never the same: different speaking rate, rate fluctuations.

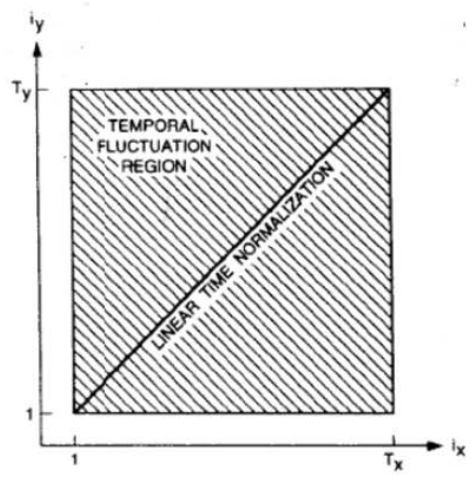
Those causes should not contribute to dissimilarities.

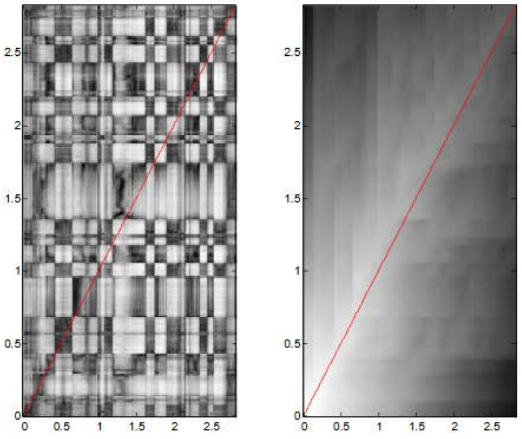
We will consider two speech patterns $X = \left(x_1, \dots, x_{T_x}\right)$ and $Y = \left(y_1, \dots, y_{T_y}\right)$ with (potentially different) lengths $T_x$ and $T_y$.

The dissimilarity between $X$ and $Y$ will be defined by a distance function $d(i_x, i_y)$

Alignment

Linear time normalization: $i_y = \frac{T_y}{T_x} i_x$ and $d(X, Y) = \sum_{i_x=1}^{T_x} d(i_x, i_y)$




We use two warping functions to $\phi_x$ and $\phi_y$ to relate indices of the two speech patterns. Those function will point to the vector that we should be comparing.

$i_x = \phi_x(k)$ and $i_y = \phi_y(k)$ for $k = 1, \dots, T$

Then a global pattern dissimilarity could be used $d_\phi(X, Y) = \sum_{k=1}^{T} d\left(\phi_x(k), \phi_y(k)\right) \frac{m(k)}{M_\phi}$

$m$ is a nonnegative path weighting the path and $M_\phi$ is a path normalizing scheme.
Choosing the path? Test all paths and choose the one giving the min distortion $d = \min_\phi d_\phi$

Time-normalization constraints:

- Endpoints constraints: $\phi_{x/y}(1) = 1$ and $\phi_{x/y}(T) = T_{x/y}$
- Monotonicity constraints: $\phi(k+1) \geq \phi(k)$ we never go back in time
- Local continuity constraints: $\phi(k+1) - \phi(k) \leq 1$

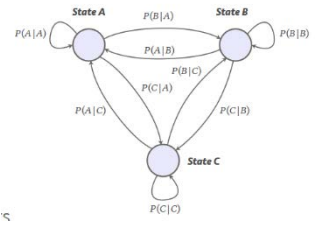
## Lectures 5, 6 & 7: Stochastic Methods of Automatic Speech Recognition

Given an Automatic Speech Recognition (ASR) system, we will calculate the probability of a word given an observation $\arg\max_W P(W|O) = \arg\max_W \frac{P(W)P(O|W)}{P(O)} = \arg\max_W P(W)P(O|W)$.
$P(W)$ is the language model and $P(O|W)$ is the acoustic model.

Stochastic: instead of comparing instances (two time series) like in DTW, we compare an instance to a trained model (it has mean and variance).

Hidden Markov Models (HMMs) are generative models, i.e. what is the probability that this model could have generated the sequence of observations we have.

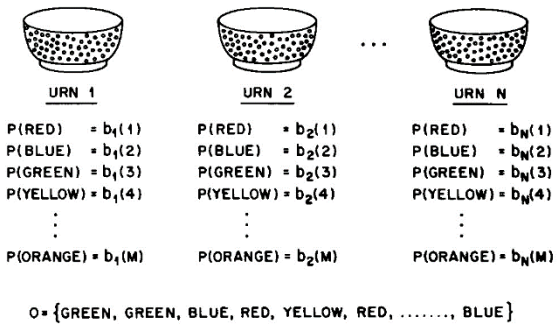
First order Markov chain: $P(q_t = j|q_{t-1} = i, q_{t-2} = k, \dots) = P(q_t = j|q_{t-1} = i)$
State transition probability $a_{ij} = P(q_t = j|q_{t-1} = i)$ this is the probability of going to state $j$ at time $t$ from a state $i$ at time $t-1$ (independent of time!)
So given $i$, $\sum_j a_{ij} = 1$




Hidden: we cannot observe the state sequence from the observations, but we can infer it.
A very complex model can describe well the situation but will not generalize.

State-transition probability distribution: $a_{ij} = P(q_{t+1} = j|q_t = i), 1 \leq i, j \leq N$
Observation symbol probability distribution: $b_j(k) = P(o_t = v_k|q_t = j), 1 \leq k \leq M$
The initial state distribution: $\pi_i = P(q_1 = i)$
The model $\lambda = (A, B, \Pi)$

Example: the urn-and-ball model



**Problem 1 – Evaluation** (estimate what the word is)

Given observations $O$ and a model $\lambda$, how to compute $P(O|\lambda)$?

We must consider every single state sequence ($N^T$ possible one) since we do not know the real state sequence.

For one particular state sequence $q = (q_1 \dots q_T)$, the probability of $O$ given $q$ is $P(O|q,\lambda) = \prod_t P(o_t|q_t,\lambda)$

Statistical independence: $P(O|q,\lambda) = b_{q_1}(o_1) \dots b_{q_T}(o_T)$

The probability of a state sequence $P(q|\lambda) = \pi_{q_1} a_{q_1 q_2} \dots a_{q_{T-1} q_T}$

Joint probability $P(O,q|\lambda) = P(O|q,\lambda)P(q|\lambda)$

Finally, we sum over all possible $q$: $P(O|\lambda) = \sum_q P(O|q,\lambda)P(q|\lambda) = \sum_{q_1,\dots,q_T} \pi_{q_1} b_{q_1}(o_1) a_{q_1 q_2} \dots a_{q_{T-1} q_T} b_{q_T}(o_T)$

This is so costly; we need a more efficient procedure!

**The forward procedure**

We consider the forward variable $\alpha_t(i) = P(o_1 \dots o_t, q_t = i|\lambda)$, i.e. the probability of partial observation sequence $o_1 \dots o_t$ and state $i$ at time $t$ given $\lambda$ to remove redundant calculations. And we solve for $\alpha_t(i)$:

1. Initialization: $\alpha_1(i) = \pi_i b_i(o_1), 1 \le i \le N$
2. Induction: $\alpha_{t+1}(j) = \left[\sum_i \alpha_t(i) a_{ij}\right] b_j(o_{t+1}), 1 \le t \le T-1$
3. Termination: $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

Benefit: we go from $2TN^T$ to $N^2T$ calculations.

**The backward procedure**

In the same manner, we can consider a backward variable $\beta_t(i) = P(o_{t+1} \dots o_T|q_t = i,\lambda)$, i.e. the probability of partial observation sequence $o_{t+1} \dots o_T$ given state $i$ at time $t$ and the model. And we solve for $\beta_t(i)$:

1. Initialization: $\beta_T(i) = 1, 1 \le i \le N$
2. Induction: $\beta_t(i) = \sum_j a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)\,, t = T-1, \dots, 1 \; and \; 1 \le i \le N$

**Problem 2 – Decoding / State assignment**

Given observations $O$ and a model $\lambda$, how to determine the state sequence $q = (q_1 \dots q_T)$ that explains $O$?

On solution is to choose states $q_t$ that are individually most likely. So we define the variable

$$\gamma_t(i) = P(q_t = i|O,\lambda) = \frac{P(O, q_t = i|\lambda)}{P(O|\lambda)} = \frac{P(O, q_t = i|\lambda)}{\sum_i P(O, q_t = i|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_i \alpha_t(i)\beta_t(i)}$$

We can solve for the individually most likely state at each time $t$:

$$q_t^* = \arg\max_i \gamma_t(i)\,, 1 \le t \le T$$

This does not consider the probability of sequences! Solution: find the single best state sequence.

Viterbi Algorithm: we define the quantity $\delta_t(i) = \max_{q_1,\dots,q_{t-1}} P(q_1 \dots q_t = i, o_1 \dots o_t|\lambda)$ which is the best score along a single path that ends in state $i$. By induction we have $\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij}\right] b_j(o_{t+1})$

1. Initialization: $\delta_1(i) = \pi_i b_i(o_1), 1 \le i \le N$ and $\psi_1(i) = 0$
2. Recursion: $\delta_t(j) = \max_i\left[\delta_{t-1}(i) a_{ij}\right] b_j(o_t)$ and $\psi_t(j) = \arg\max_i\left[\delta_{t-1}(i) a_{ij}\right]$
3. Termination: $P^* = \max_i \delta_T(i)$ and $q^* = \arg\max_i \delta_T(i)$
4. Path Backtracking: $q_t^* = \psi_{t+1}(q_{t+1}^*), t = T-1, \dots, 1$

**Problem 3 – Learning / Optimization / Training / Estimation**

How to adjust/estimate the model parameters $(A, B, \Pi)$ to maximize $P(O|\lambda)$?

Problem: no closed form analytical solution.

We define the variable $\zeta_t(i,j)$ the probability of being in state $i$ at $t$ and at state $j$ at $t+1$.

$$\zeta_t(i,j) = P(q_t = i, q_{t+1} = j|O,\lambda) = \frac{P(q_t = i, q_{t+1} = j, O|\lambda)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}$$

We can write $\gamma_t(i) = \sum_j \zeta_t(i,j)$ the probability of being in state $i$ at time $t$.

$\sum_t \gamma_t(i)$ = a quantity that describes the expected number of times state $i$ was visited.

$\sum_t \zeta_t(i,j)$ = a quantity that describes the expected number of transitions from $i$ to $j$.

Estimate the model parameters:

$\bar{\pi}_i$ = expected frequency of in state $i$ at time $1 = \gamma_1(i)$

$\bar{a}_{ij}$ = expected number of transition from $i$ to $j$ / expected number of transition from $i = \frac{\sum_t \zeta_t(i,j)}{\sum_t \gamma_t(i)}$

$\bar{b}_j(k)$ = expected number of times in $j$ with $v_k$ / expected number of times in $j = \frac{\sum_{t,o_t=v_k} \gamma_t(i)}{\sum_t \gamma_t(i)}$