Speech recognition – an introduction

Jan Černocký FIT VUT Brno, `cernocky@fit.vutbr.cz`

April 11, 2005

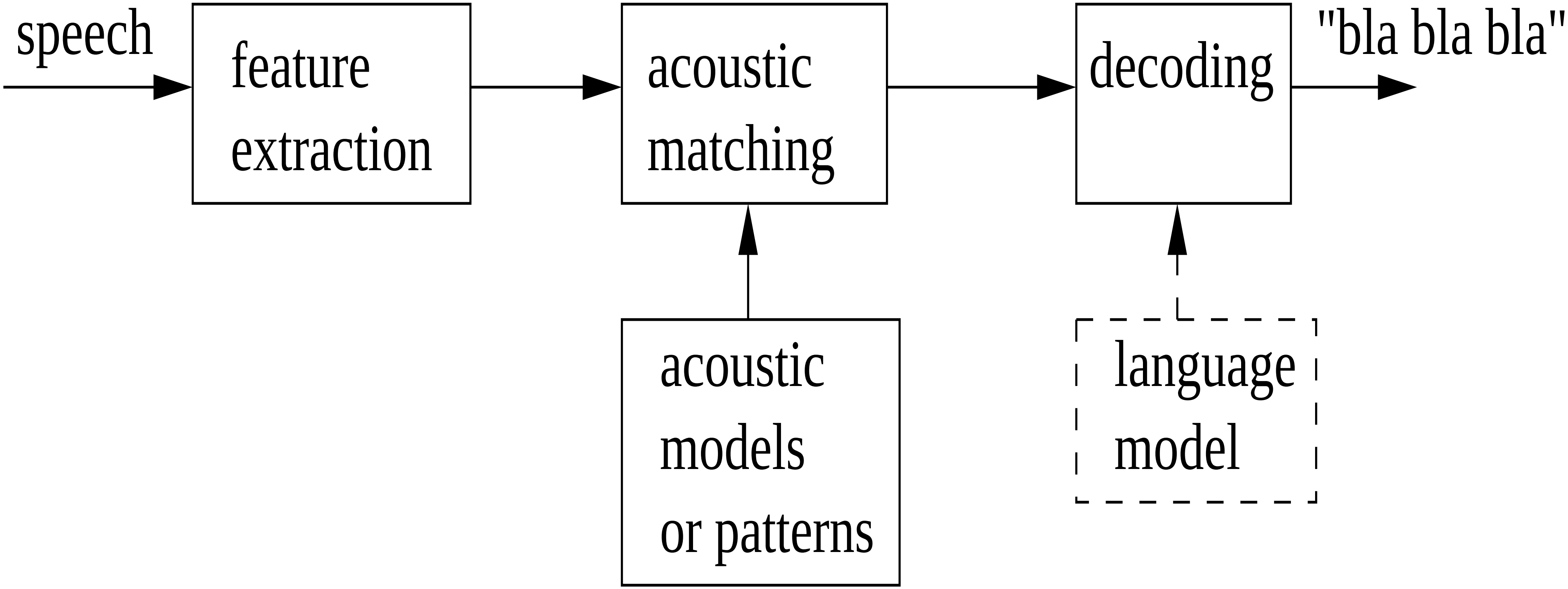
Speech Recognition — an Introduction

The task: for an unknown signal, determine what was said.

Classification:

- isolated words – commanding of mobile phones by voice, very limited vocabulary, processing just one single word.
- connected words (limited vocabulary) – for example entering telephone or credit card numbers by voice. The recognition is driven by a recognition network or a simple grammar.
- continuous speech with large vocabulary (large vocabulary continuous speech recognition LVCSR) – the hardest, needs info on acoustics, but also about the structure of language (language model) and pronunciations (pronunciation dictionary). Smaller units than words (can't train 60k words . . .) - phonemes, context-dependent phonemes.

Structure of the recognizer



Plan of this talk

- Feature extraction - LPCC and Mel-frequency cepstral coefficients.
- The variability everywhere.
- Coping with the variability by template-matching (DTW - dynamic time warping).
- Coping with the variability by statistic modeling (Hidden Markov Models).

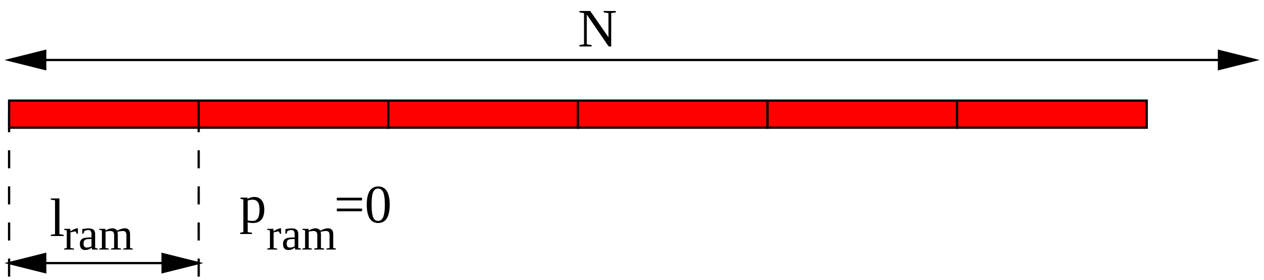
FEATURES FOR SPEECH RECOGNITION

the aim of feature extraction is to select a limited number of parameters (features) describing speech in each frame, that would be:

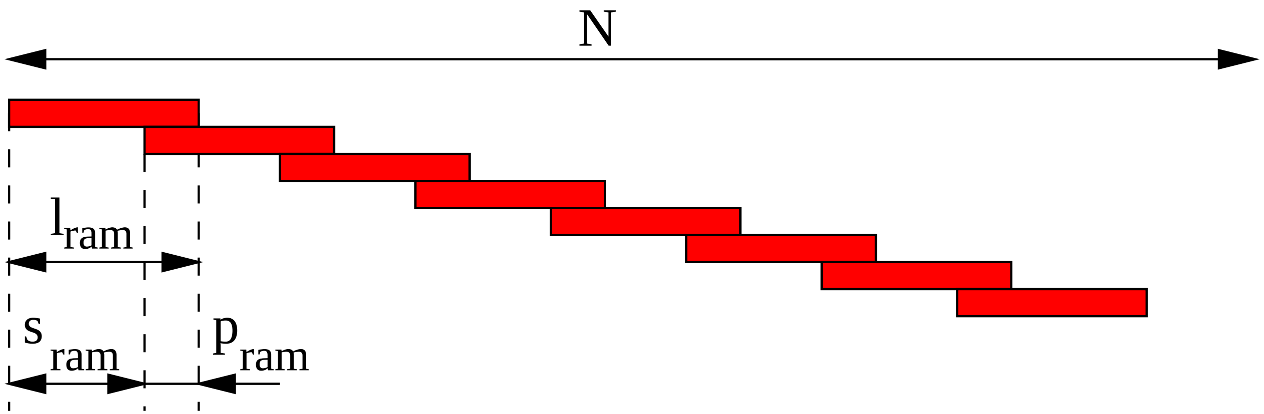
- representative for phonetic content but invariant for other sources of information (pitch, phase).
- the features should fit well with the recognizer's needs, e.g. the distance of feature vectors should have some physical sense or they should be de-correlated.

Global features ? Not, in frames.

Non-overlapping frames



Overlapping frames



Cepstral analysis

$$\ln G(f) = \sum_{n=-\infty}^{+\infty} c(n) e^{-j2\pi f n} \tag{1}$$

Values $c(n)$ are called **cepstral coefficients**. As $G(f)$ is an even function, such coefficients are real and even:

$$c(n) = c(-n) \tag{2}$$

The sum in Eq. 1 is a definition of DFT, so that:

$$c(n) = \mathcal{F}^{-1}\left[\ln G(f)\right] \tag{3}$$

where $\mathcal{F}^{-1}$ stands for inverse Fourier transform. According to the estimation of $G(f)$, two methods exist for the estimation of cepstrum:

DFT-cepstrum

PSD is estimated using DFT:

$$c(n) = \mathcal{F}^{-1}\left\{\ln\left|\mathcal{F}[s(n)]\right|^2\right\}, \tag{4}$$

DFT is usually implemented using the fast algorithm FFT. As n is again the discrete time (samples), it is called "quefrency". In case of cepstral analysis of two convoluted signals:

$$s(n) = e(n) \star h(n), \tag{5}$$

where $e(n)$ denotes excitation and $h(n)$ impulse response of filter, the spectrum of convolution is given by a product of original spectra:

$$S(f) = E(f)H(f) \text{ and therefore } |S(f)|^2 = |E(f)|^2\ |H(f)|^2. \tag{6}$$

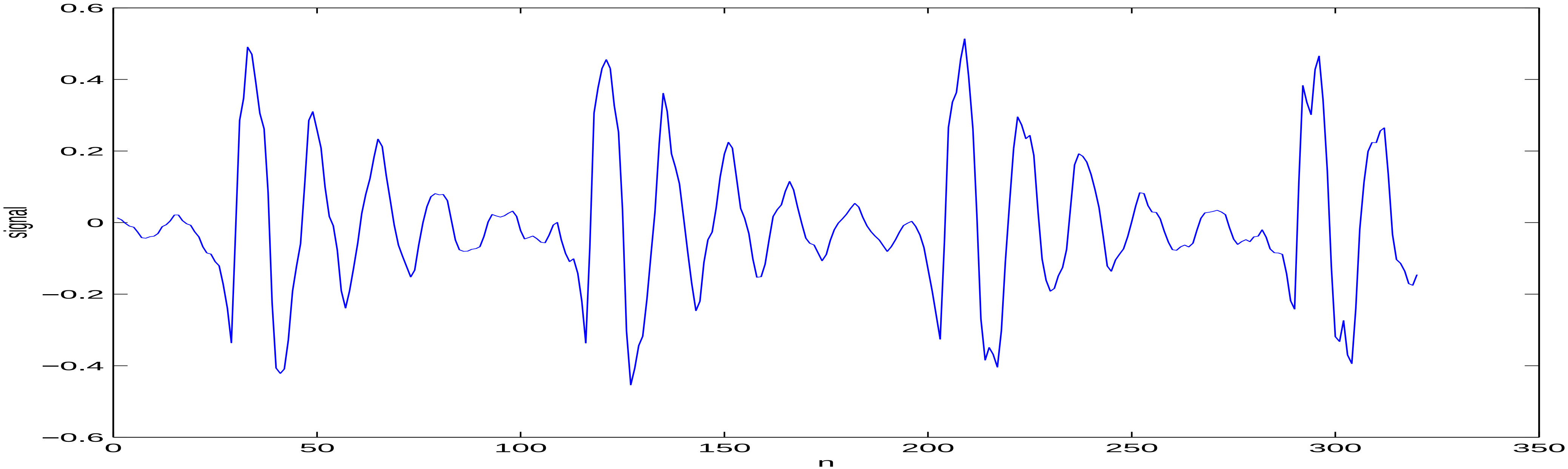
Inverse Fourier transform is linear:

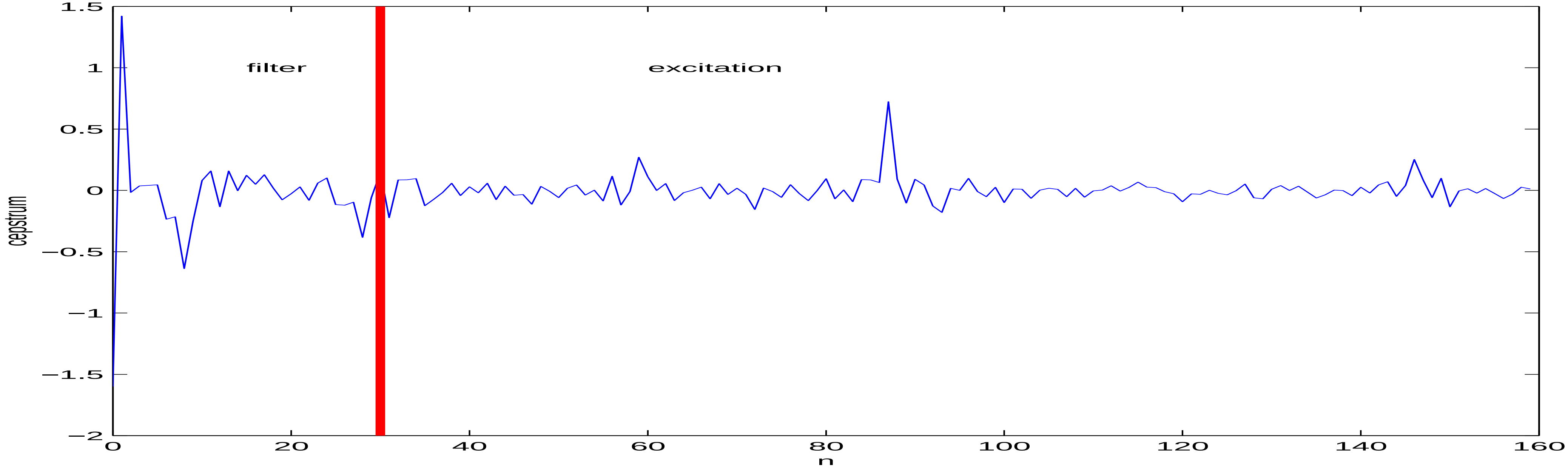
$$\mathcal{F}^{-1}(a+b) = \mathcal{F}^{-1}(a) + \mathcal{F}^{-1}(b).$$

and we obtain:

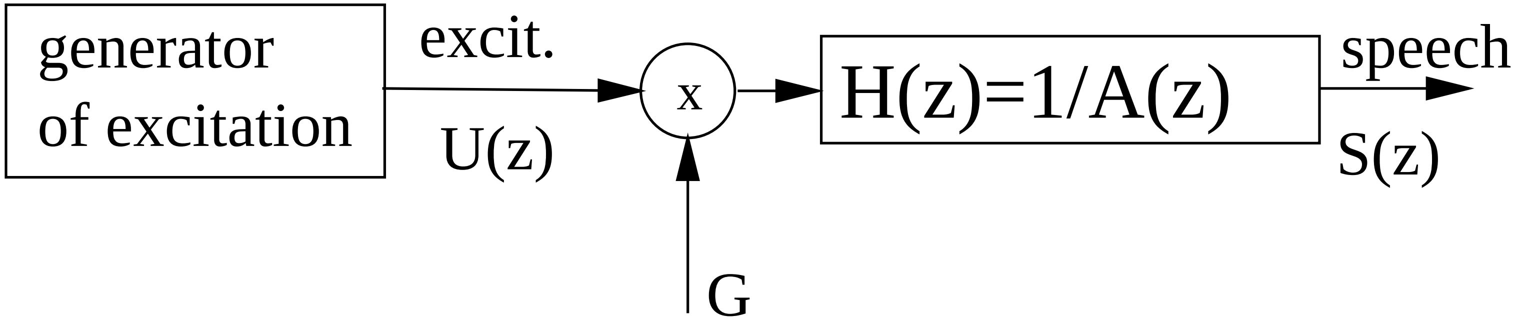
$$\begin{aligned} c(n) &= \mathcal{F}^{-1}\left\{\ln[|E(f)|^2 \; |H(f)|^2]\right\} = \mathcal{F}^{-1}\left\{\ln|E(f)|^2 + \ln|H(f)|^2\right\} = & (7) \\ &= \mathcal{F}^{-1}\left\{\ln|E(f)|^2\right\} + \mathcal{F}^{-1}\left\{\ln|H(f)|^2\right\} = c_e(n) + c_h(n) & (8) \\ & & (9) \end{aligned}$$

The convolution has become **summation**. In case coefficients $c_e(n)$ and $c_h(n)$ are not mixed on quefrency axis, they may be separated using a simple windowing.


signal
n
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
0
50
100
150
200
250
300
350


cepstrum
filter
excitation
n
1.5
1
0.5
0
−0.5
−1
−1.5
−2
0
20
40
60
80
100
120
140
160

LPC-cepstrum



Using estimation of PSD from LPC:

$$\hat{G}_L(f) = \left| \frac{G}{A(z)} \right|^2_{z=e^{j2\pi f}}, \tag{10}$$

where G is gain of "synthesis" filter and $A(z)$ is a polynomial of order P. LPC-cepstrum (LPCC) is then:

$$c(n) = \mathcal{F}^{-1}[\hat{G}_{LPC}(f)] \tag{11}$$

Zeroth LPC-cepstral coefficient carries the information about the energy of speech:

$$c(0) = \ln G^2. \tag{12}$$

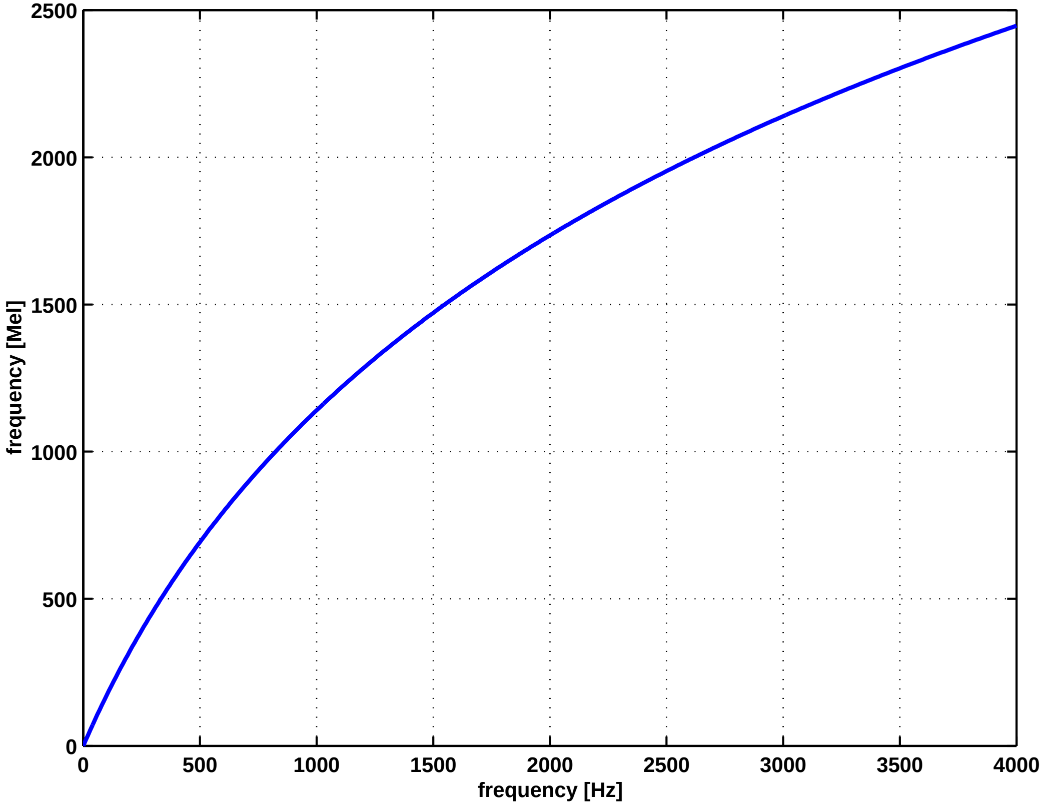
The other coefficients can be computed using a recursion:

$$
\begin{aligned}
c(n) &= -a_n - \frac{1}{n}\sum_{k=1}^{n-1} k c_k a_{n-k} \quad \text{for} \quad 1 \leq n \leq P \\
c(n) &= -\frac{1}{n}\sum_{k=1}^{n-1} k c_k a_{n-k} \qquad\quad \text{for} \quad n > P
\end{aligned}
\tag{13}
$$

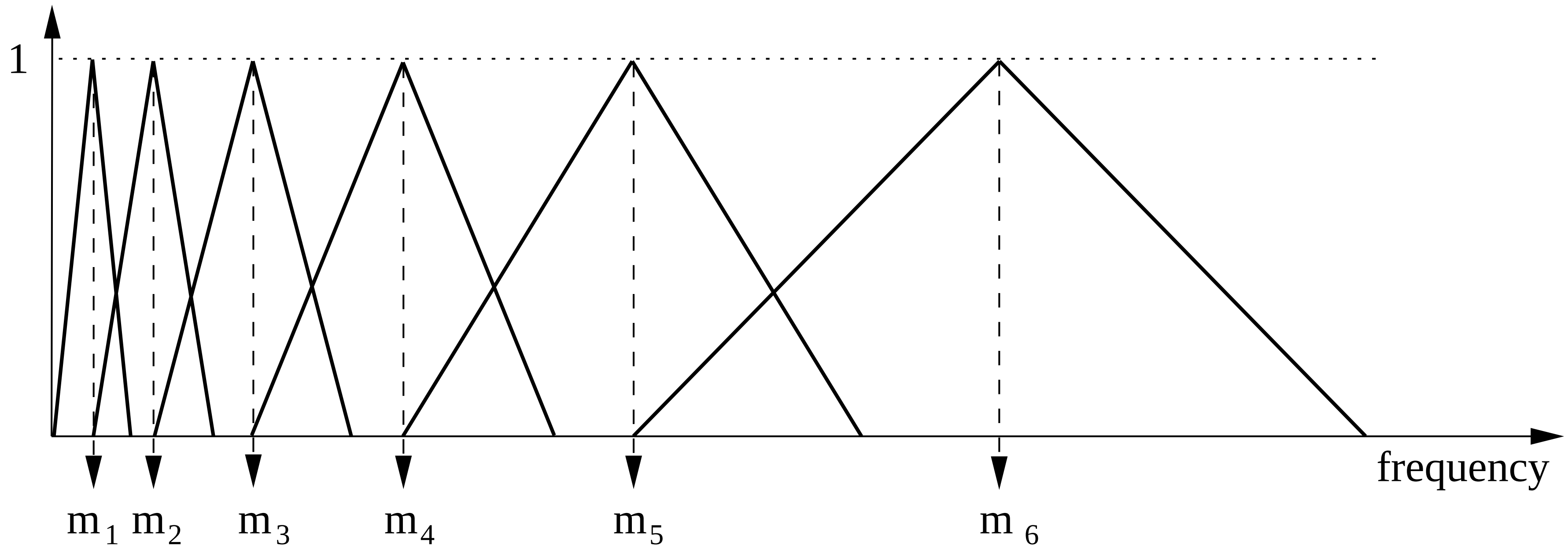
Mel-frequency cepstral coefficients

Non-linear frequency transform:

$$F_{Mel} = 2959 \log_{10}(1 + \frac{F_{Hz}}{700}) \qquad (14)$$

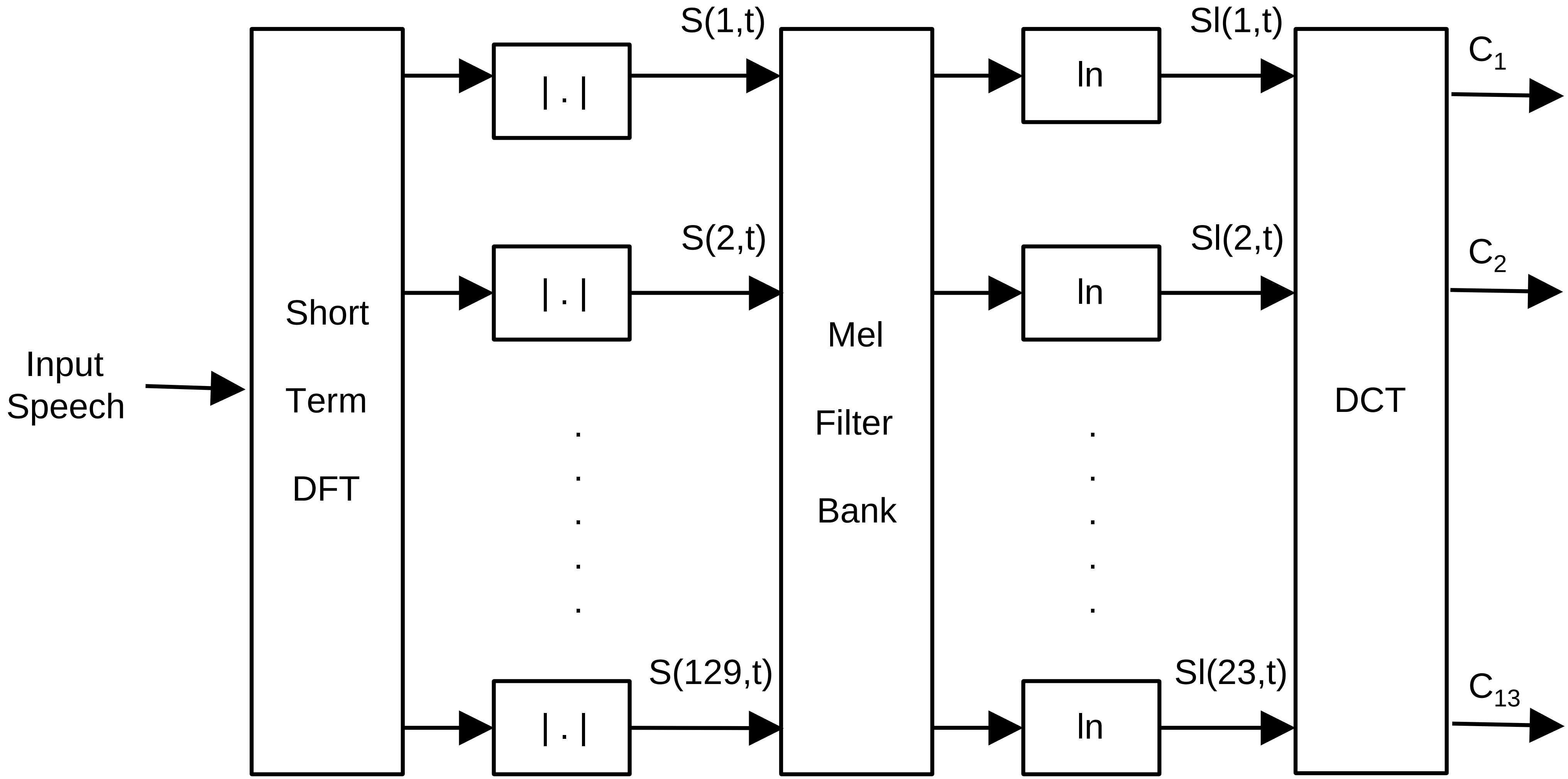


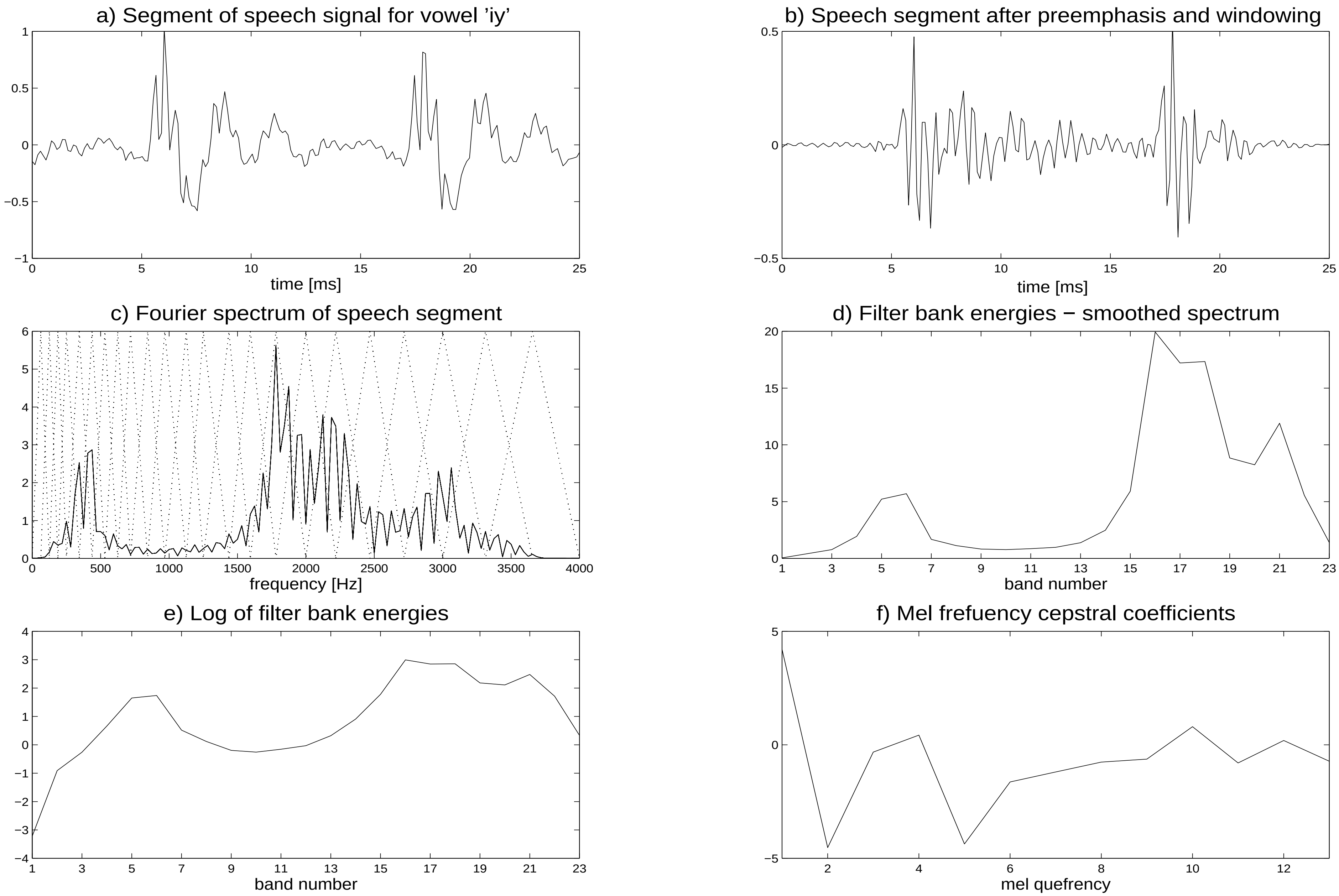
Linear placing of filters on Mel-axis results in non-linear placement on the frequency axis:



Energies of signals from filters are computed by “binning” the powers of DFT. The inverse Fourier transform can be computed using discrete cosine transform (DCT):

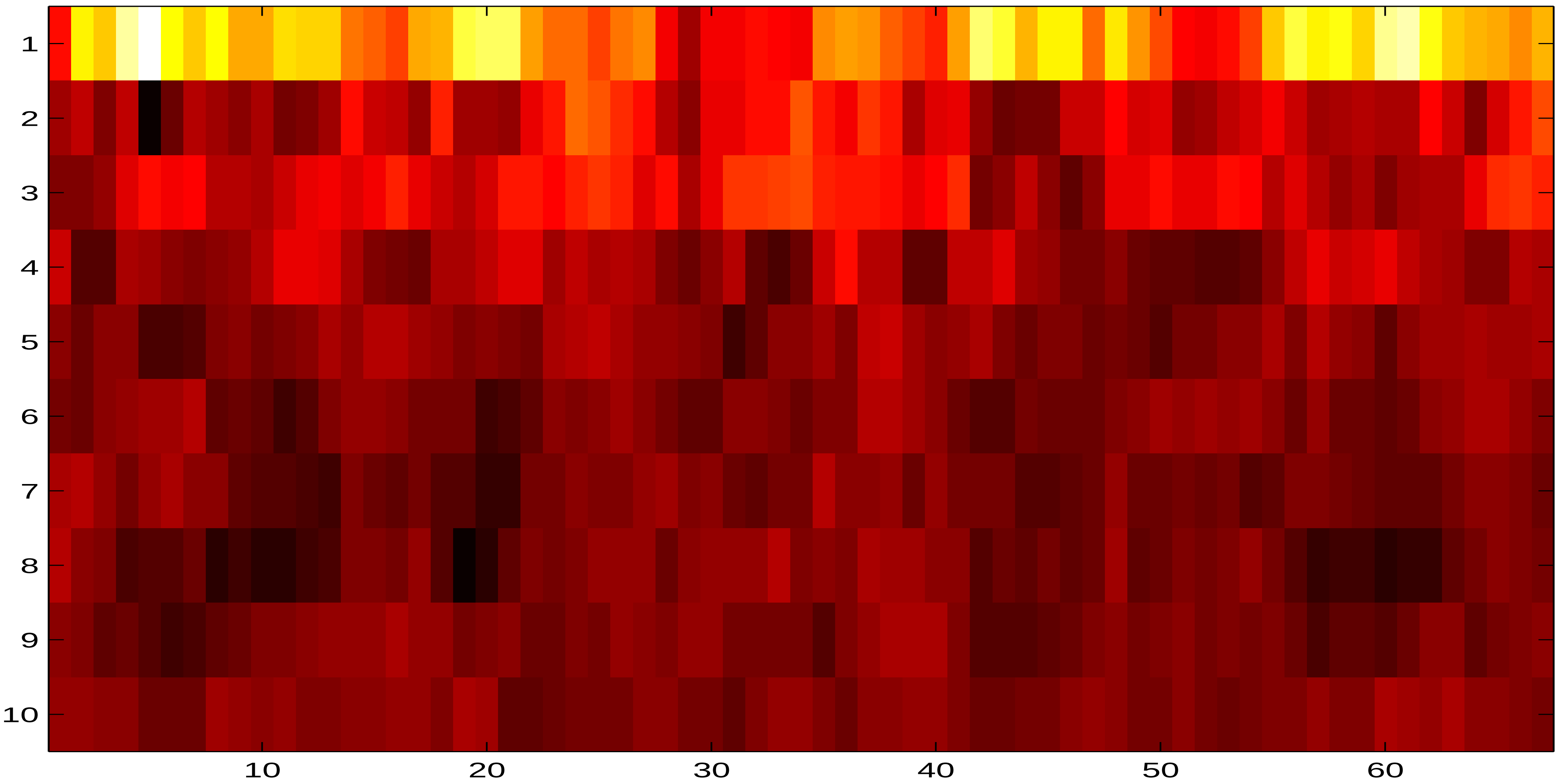
$$c_{mf}(n) = \sum_{i=1}^{K} \log m_k \cos\left[n(k-0.5)\frac{\pi}{K}\right] \qquad (15)$$


Input Speech
Short Term DFT
| . |
S(1,t)
S(2,t)
S(129,t)
Mel Filter Bank
ln
Sl(1,t)
Sl(2,t)
Sl(23,t)
DCT
C_1
C_2
C_{13}


a) Segment of speech signal for vowel 'iy'
time [ms]
b) Speech segment after preemphasis and windowing
time [ms]
c) Fourier spectrum of speech segment
frequency [Hz]
d) Filter bank energies − smoothed spectrum
band number
e) Log of filter bank energies
band number
f) Mel frefuency cepstral coefficients
mel quefrency

So what do we get ?

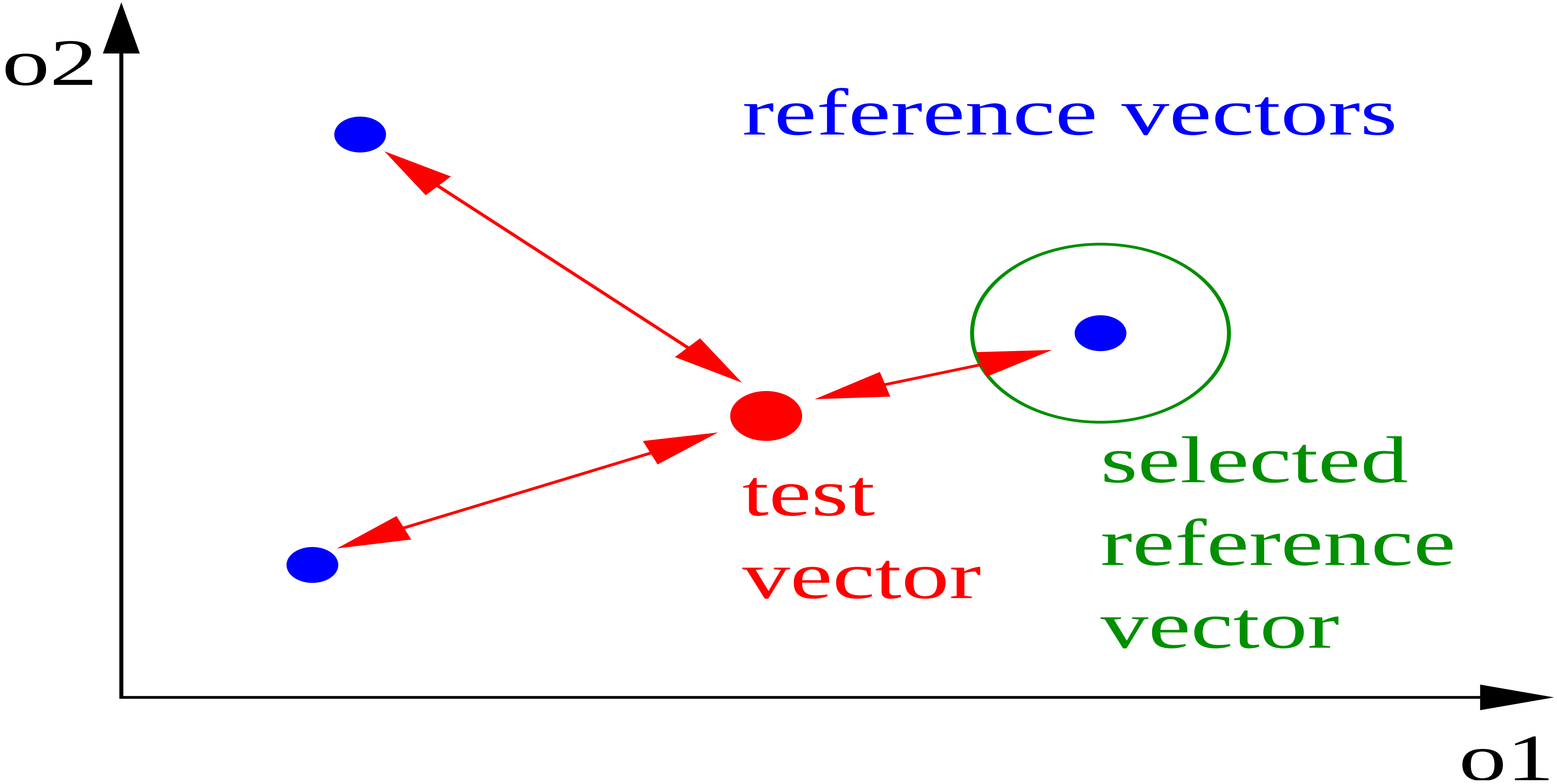
A sequence of vectors: $\mathbf{O} = [\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(T)]$

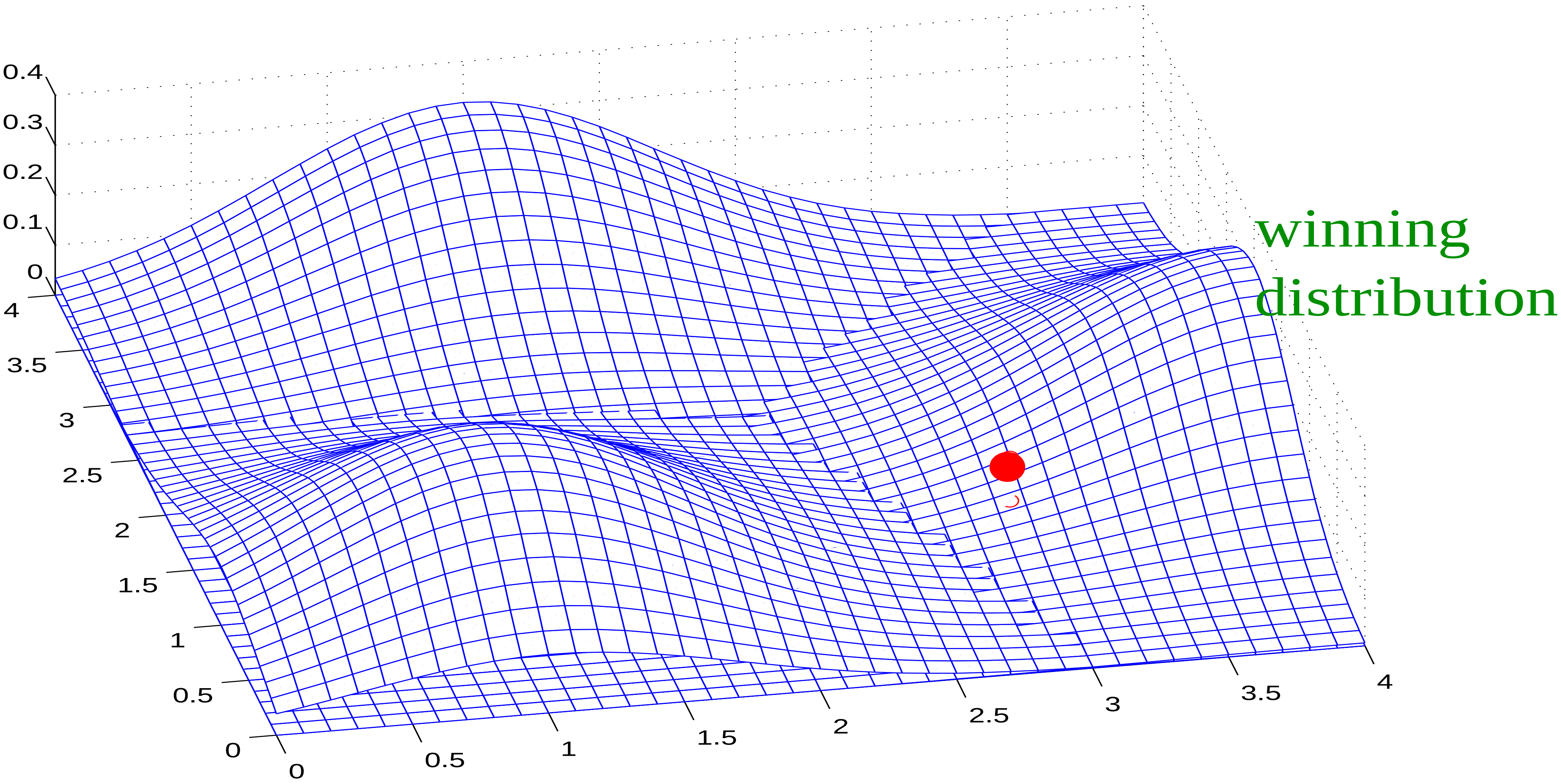


ACOUSTIC MATCHING — THE VARIABILITY EVERYWHERE

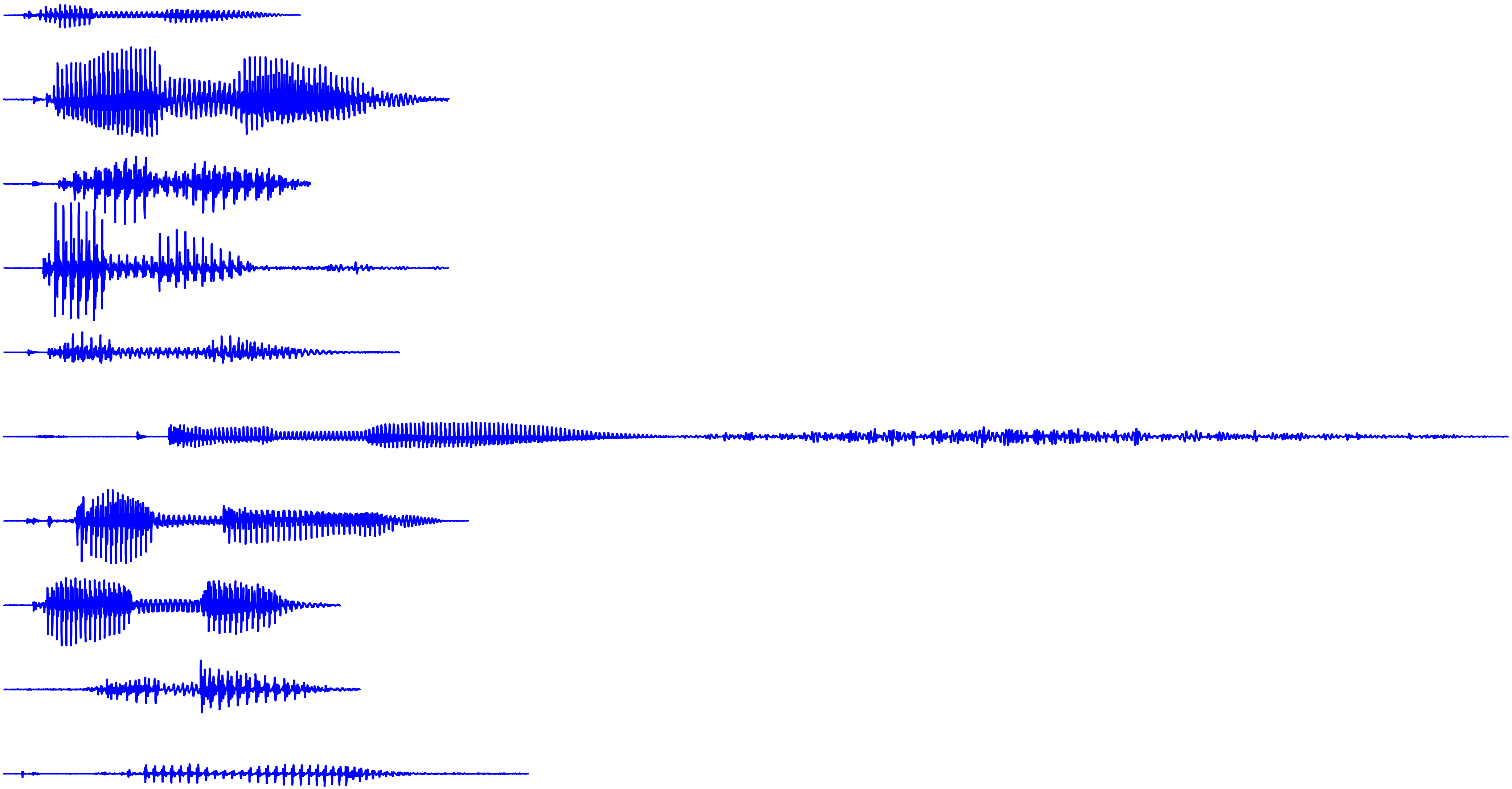
1. feature space - even one human NEVER says something the same way. ⇒ the vectors of parameters are **always different**. The methods good for text won't work ⇒ What to do ?

1. Measure distances between vectors.
2. Model vectors statistically.

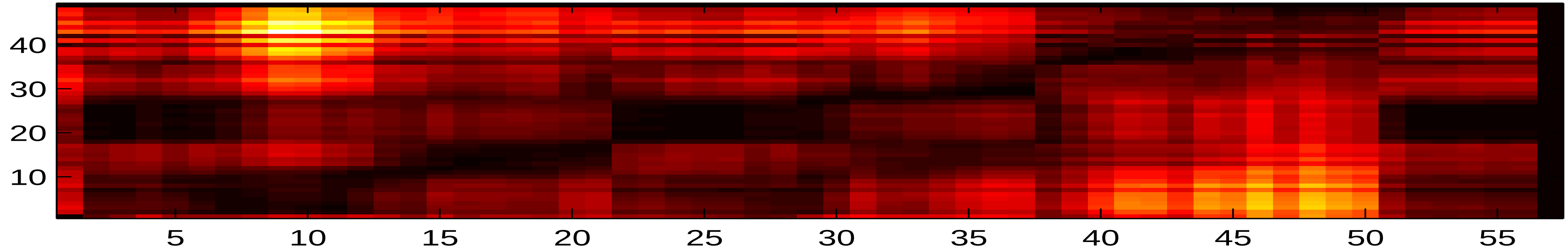

o2
reference vectors
test
vector
selected
reference
vector
o1

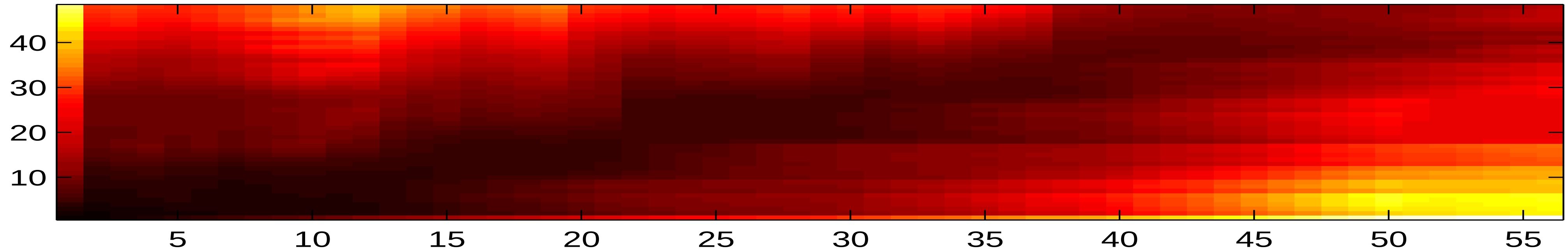

0.4
0.3
0.2
0.1
0
4
3.5
3
2.5
2
1.5
1
0.5
0
0
0.5
1
1.5
2
2.5
3
3.5
4
winning
distribution

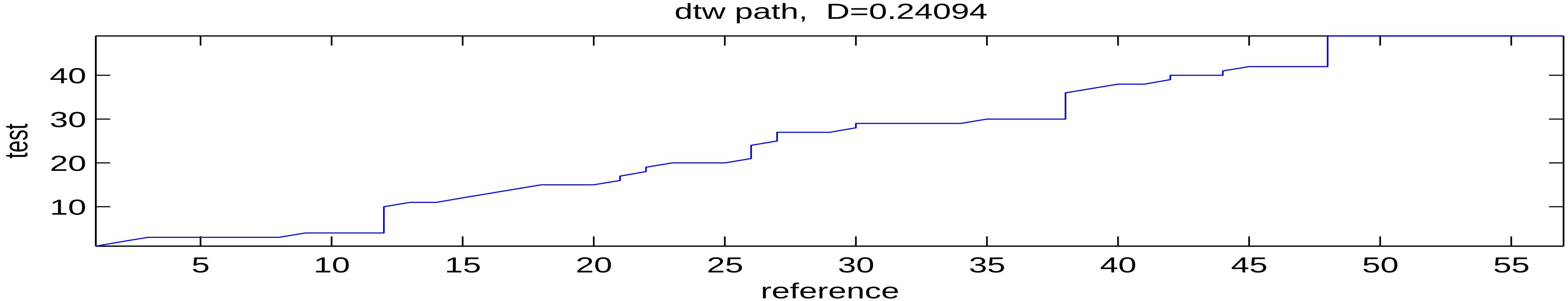
2. timing – people NEVER say the same thing with the same timing.



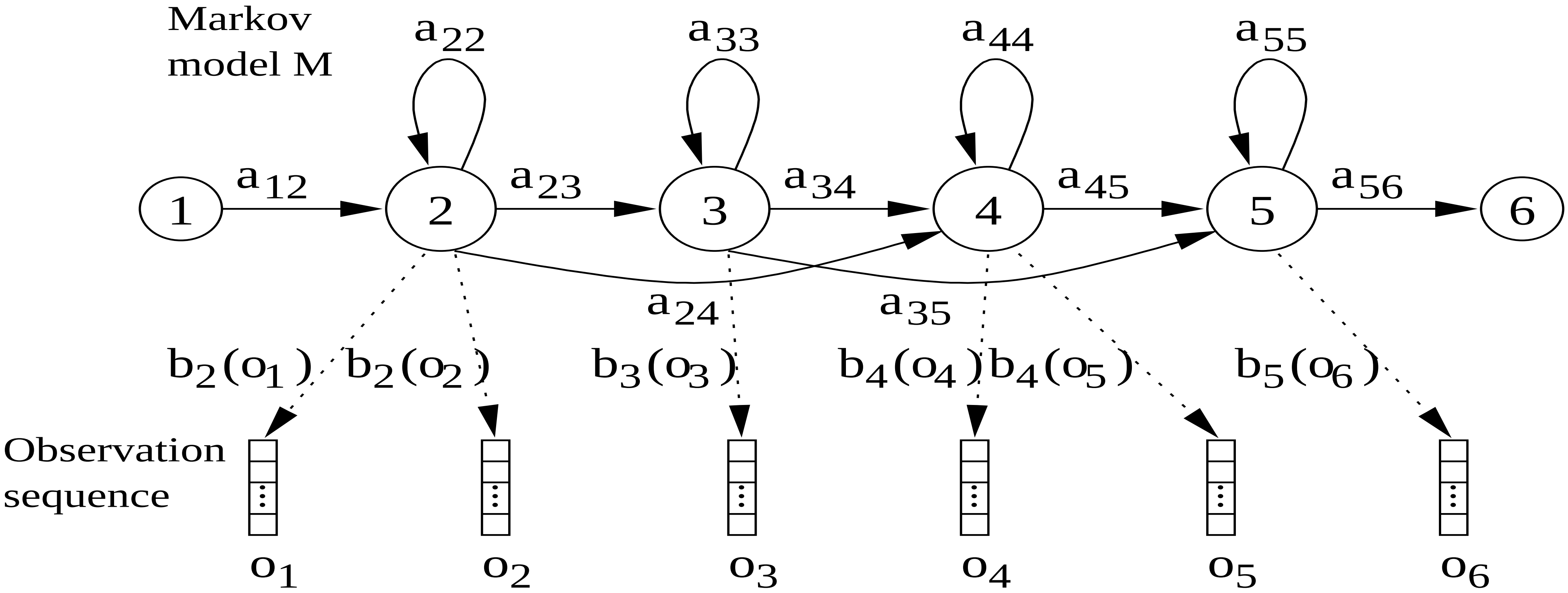
Timing No.1 - Dynamic time warping - the path







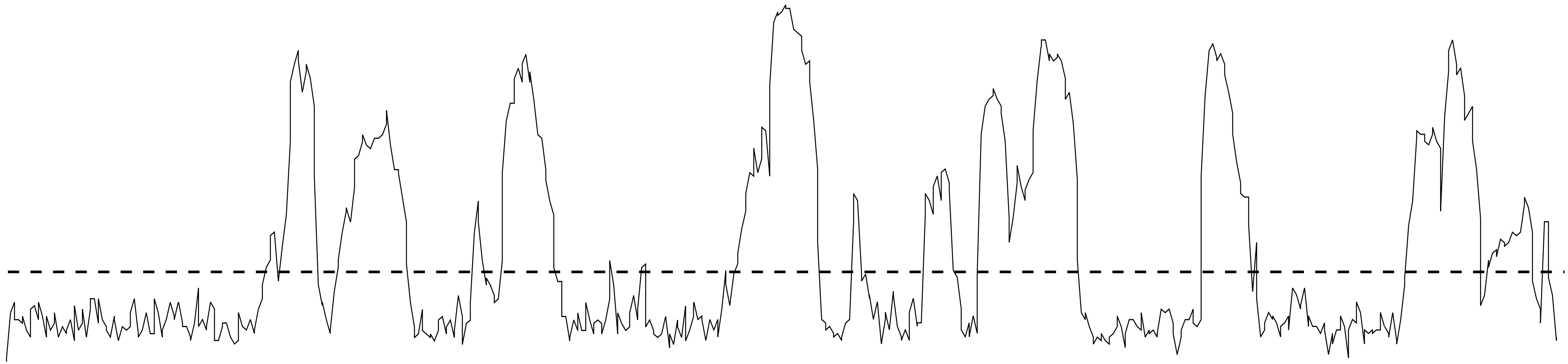
Timing No.2 - Hidden Markov models - the state sequence.



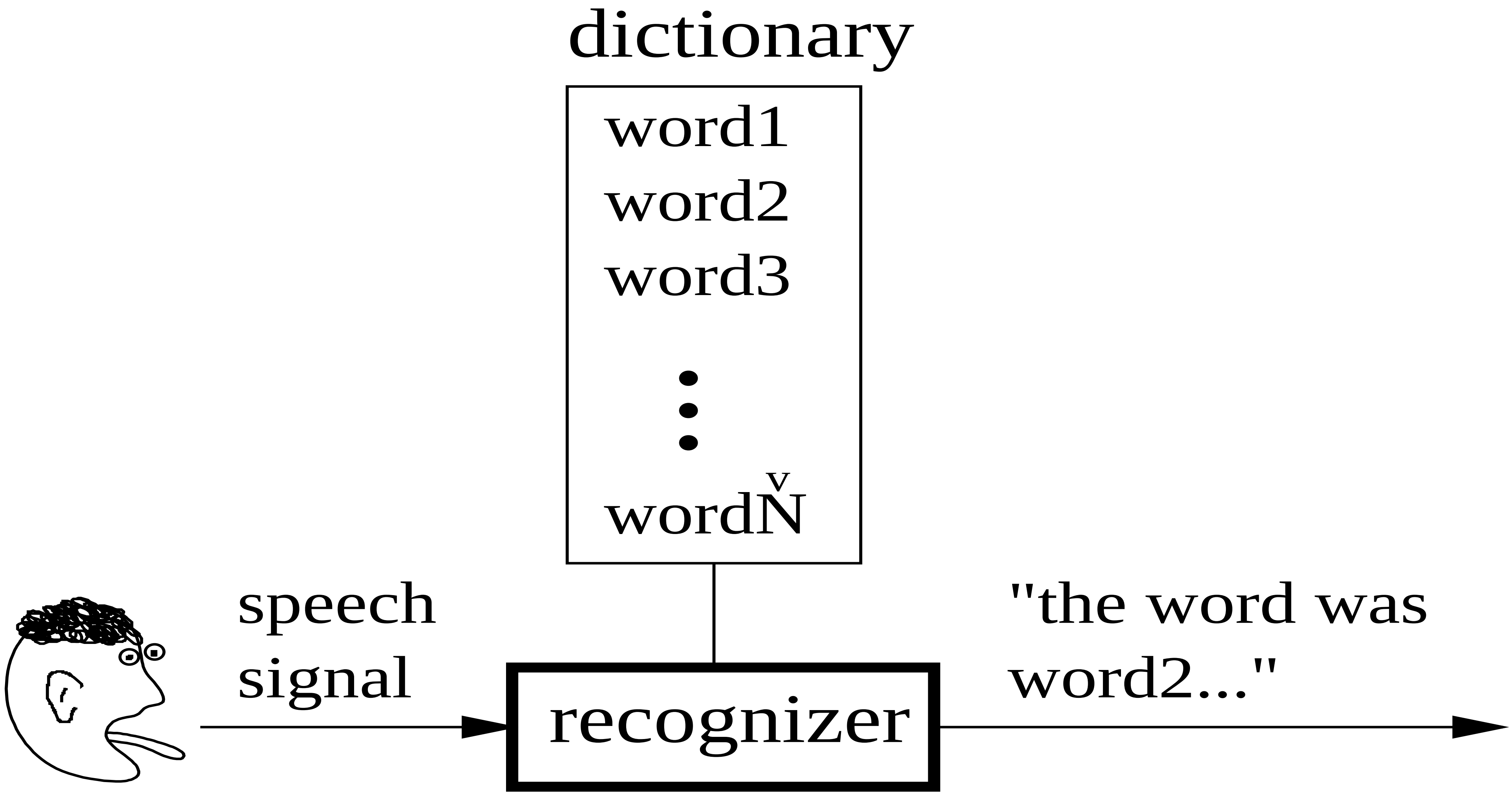
RECOGNITION OF ISOLATED WORDS BASED ON DTW

Where are these isolated words ?

- push-to-talk. . .
- voice activity detection - for example based on the energy:



How it should work:



- there are **reference** matrices of features for words we want to recognize.

$$\mathbf{R}_1 \ldots \mathbf{R}_{\check{N}}$$

- There is a **test** matrix of features at the input of the recognizer: $\mathbf{O}$
- we want to determine, to which reference the test belongs.

If only the words had only 1 vector...

$$d(\mathbf{o}, \mathbf{r}_i) = \sqrt{\sum_{k=1}^{P} |o(k) - r_i(k)|^2}.$$

Selection according to minimum distance.

But the words **don't have just 1 vector**: Need to determine the *distance* (or *similarity*) of reference sequence of vectors (length R):

$$\mathbf{R} = [\mathbf{r}(1), \ldots, \mathbf{r}(R)] \tag{16}$$

with the test sequence of vectors (length T):

$$\mathbf{O} = [\mathbf{o}(1), \ldots, \mathbf{o}(T)] \tag{17}$$

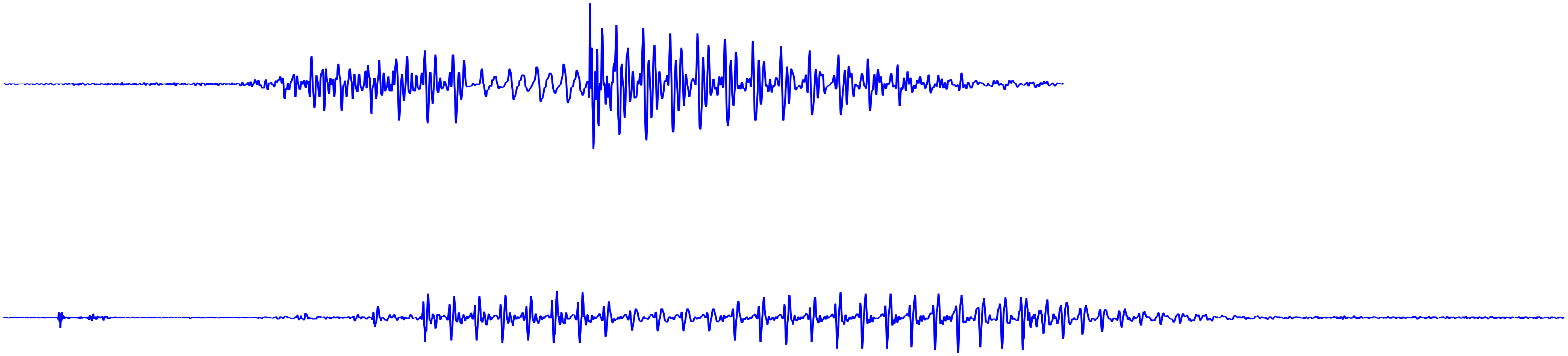
Sum the distances of all vectors ? Which vectors ?! The words have **never** the same lengths ! $R \neq T$.

Linear alignment ?

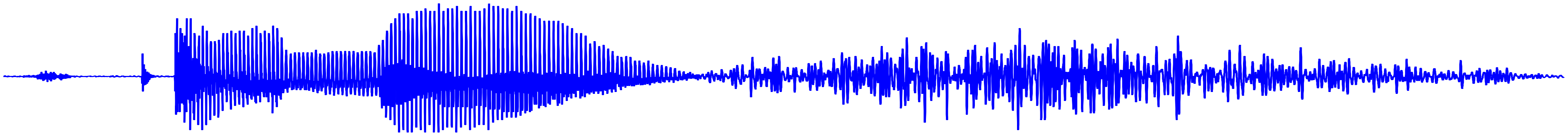
$$D(\mathbf{O},\mathbf{R}) = \sum_{i=1}^{R} d[\mathbf{o}(w(i)),\mathbf{r}(i)] \tag{18}$$

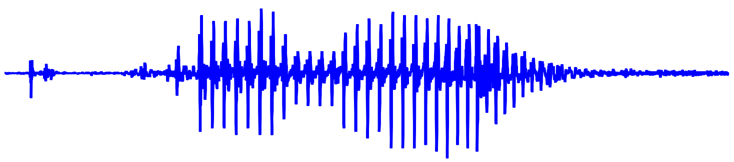
where $w(i)$ is defined to have a linear alignment...

Here it's going to work...



. . . but not here (error of VAD):





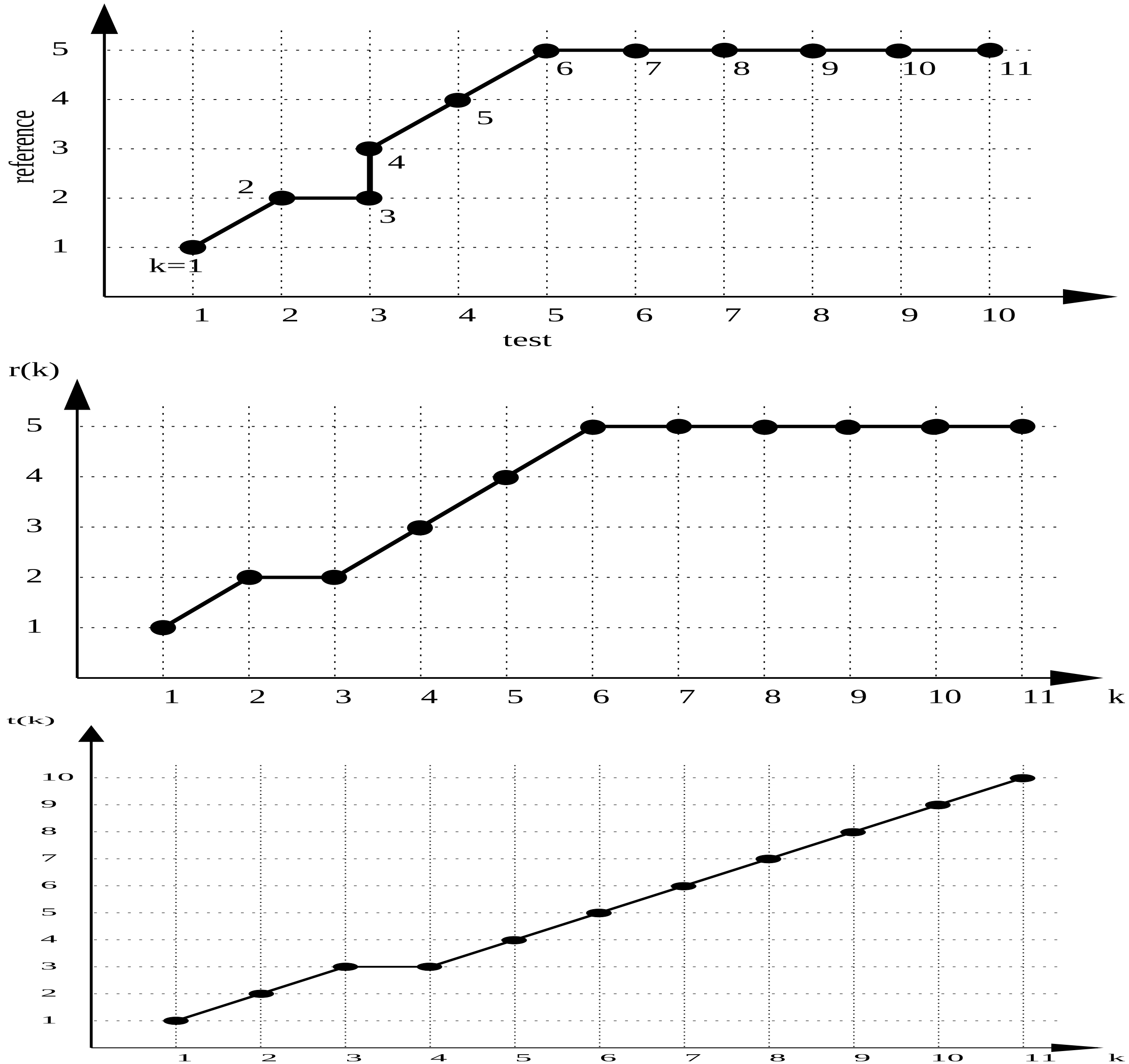
It is much better if the alignment is driven directly by the distance of different vectors $\Rightarrow$ **Dynamic time warping**.

define a general time variable k and introduce *two* transform functions:

- $r(k)$ for the reference sequence.
- $t(k)$ for the test sequence.

We can then imagine the alignment of individual vectors along a *path*. We denote the number of steps in the path K.

The reference will be shown on the vertical axis, the test on the horizontal one. From this path, functions indexing the two sequences: $r(k)$ and $t(k)$ can be derived:



Each path C is determined by its length K_C and values of functions $r_C(k)$ and $t_C(k)$. Along this path, the distance of sequences $\mathbf{O}$ and $\mathbf{R}$ is written:

$$D_C(\mathbf{O},\mathbf{R}) = \frac{\sum_{k=1}^{K_C} d[\mathbf{o}(t_C(k)), \mathbf{r}(r_C(k))] W_C(k)}{N_C} \tag{19}$$

where $d[\mathbf{o}(\cdot), \mathbf{r}(\cdot)]$ is distance of two vectors, $W_C(k)$ is a weight corresponding to the k-th step of the path and N_C is a normalization factor depending on the weights.

The DTW-distance of $\mathbf{O}$ and $\mathbf{R}$ is defined as the *minimum distance* among all possible paths:

$$D(\mathbf{O},\mathbf{R}) = \min_{\{C\}} D_C(\mathbf{O},\mathbf{R}). \tag{20}$$

It is necessary to:

1. determine allowed values of functions $r(k)$ a $t(k)$. It is not possible, that the path returns, “jumps” over several vectors, etc.
2. define weights and the normalization factor.
3. develop an algorithm that will compute $D(\mathbf{O},\mathbf{R})$ efficiently.

Path limitations

1. **Beginning and end points**

$$\left.\begin{array}{l} r(1)=1 \\ t(1)=1 \end{array}\right\} \text{beginning} \qquad \left.\begin{array}{l} r(K)=R \\ t(K)=T \end{array}\right\} \text{end} \tag{21}$$

2. **Local continuity and local steepness**

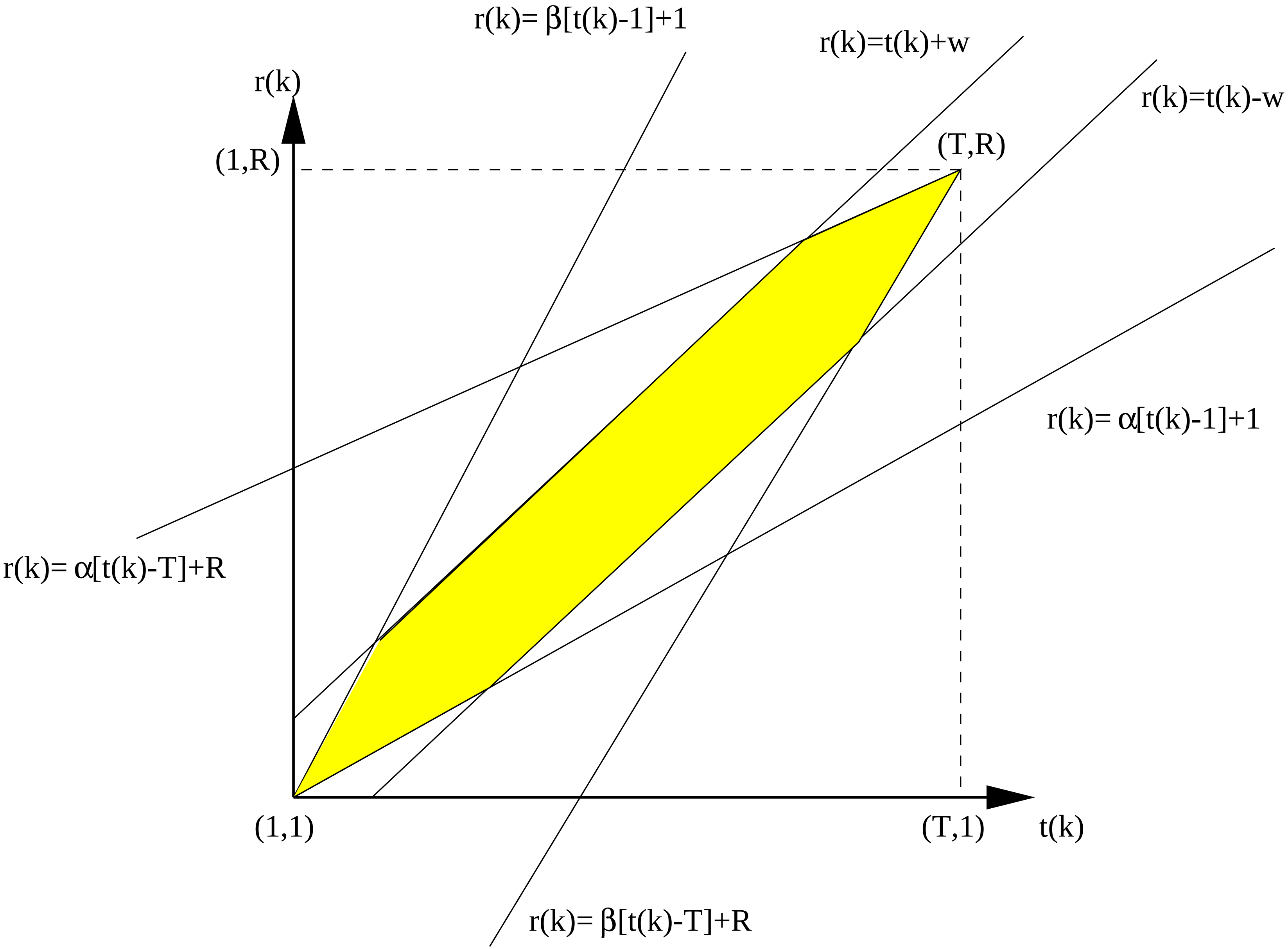
$$\begin{array}{c} 0 \leq r(k)-r(k-1) \leq R^{\star} \\ 0 \leq t(k)-t(k-1) \leq T^{\star} \end{array} \tag{22}$$

in practice, $R^{\star}, T^{\star}$=1,2,3.

- $R^{\star}, T^{\star}$=1: Each vector must be taken at least once. $r(k)=r(k-1)$ means, that the vector is repeated.
- $R^{\star}, T^{\star}>1$: Vector(s) can be skipped.

area using lines:

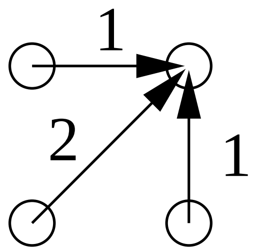
Those conditions limit the maximum "steepness" or "flatness" of the DTW path:



Weighting function.

The weighting function $W(k)$ depends on the local "shift" or "step" of the path:

- **type a)** symmetric: $W_a(k) = [t(k) - t(k-1)] + [r(k) - r(k-1)]$.



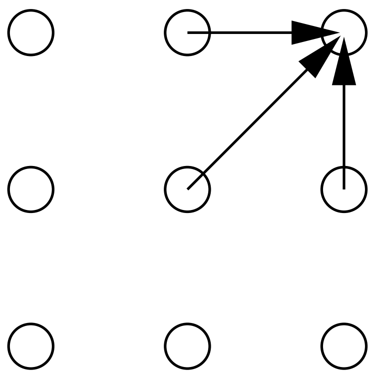
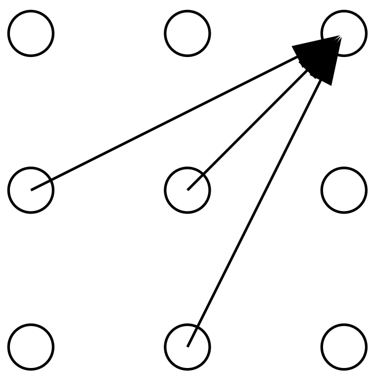
Normalization factor

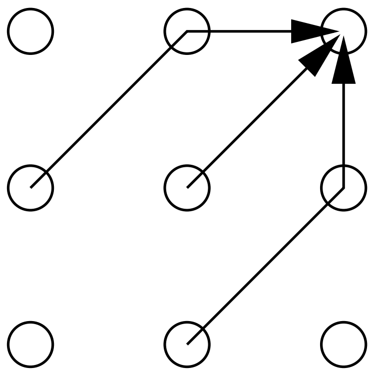
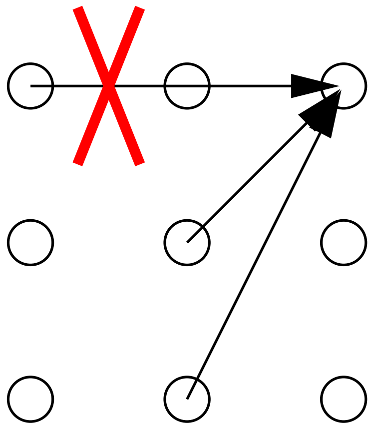
$$N = \sum_{k=1}^{K} W(k) \tag{23}$$

For weighting function of type a), the normalization factor is:

$$N_a = \sum_{k=1}^{K} [t(k) - t(k-1) + r(k) - r(k-1)] = t(K) - t(0) + r(K) - r(0) = T + R \tag{24}$$

Local limitations of the path

Type DTW		α	β	Type $w(k)$	$g(n,m)$
I.		0	∞	a	$\min\left\{\begin{array}{l} g(n,m-1)+d(n,m) \\ g(n-1,m-1)+2d(n,m) \\ g(n-1,m)+d(n,m) \end{array}\right\}$
				d	$\min\left\{\begin{array}{l} g(n,m-1)+d(n,m) \\ g(n-1,m-1)+d(n,m) \\ g(n-1,m)+d(n,m) \end{array}\right\}$
II.		$\frac{1}{2}$	2	a	$\min\left\{\begin{array}{l} g(n-1,m-2)+3d(n,m) \\ g(n-1,m-1)+2d(n,m) \\ g(n-2,m-1)+3d(n,m) \end{array}\right\}$
				d	$\min\left\{\begin{array}{l} g(n-1,m-2)+d(n,m) \\ g(n-1,m-1)+d(n,m) \\ g(n-2,m-1)+d(n,m) \end{array}\right\}$

III.		$\frac{1}{2}$	2	a	$\min\left\{\begin{array}{l} g(n-1,m-2)+2d(n,m-1)+d(n,m) \\ g(n-1,m-1)+2d(n,m) \\ g(n-2,m-1)+2d(n-1,m)+d(n,m) \end{array}\right\}$
IV.		$\frac{1}{2}$	2	b1	$\min\left\{\begin{array}{l} g(n-1,m)+kd(n,m) \\ g(n-1,m-1)+d(n,m) \\ g(n-1,m-2)+d(n,m) \end{array}\right\}$ where $k=1$ for $r(k-1)\neq r(k-2)$ $k=\infty$ for $r(k-1)=r(k-2)$

Efficient computation of $D(\mathbf{O}, \mathbf{R})$

The computation of the minimum distance

$$D(\mathbf{O}, \mathbf{R}) = \min_{\{C\}} D_C(\mathbf{O}, \mathbf{R}). \tag{25}$$

is simple provided that the normalization factor N_C does not depend on the path. We can then write:

$$N_C = N \quad \text{for} \quad \forall C$$

This fortunately holds for most cases, so that:

$$D(\mathbf{O}, \mathbf{R}) = \frac{1}{N} \min_{\{C\}} \sum_{k=1}^{K_C} d[\mathbf{o}(t_C(k)), \mathbf{r}(r_C(k))] \tag{26}$$

The procedure:

1. define a *grid of local distances* $\mathbf{d}$ of dimensions $T \times R$. Fill it with distances of reference and test vectors each-to-each.

2. define a *grid of partial cummulated distances* $\mathbf{g}$. On contrary to grid $\mathbf{d}$, $\mathbf{g}$ will have an extra 0-th line and 0-th column, which will be initialized to:

$$g(0,0)=0, \quad \text{and} \quad g(0,m\neq 0)=g(n\neq 0,0)=\infty.$$

3. For each point $[m,n]$, the partial cummulated distance will be computed:

$$g(m,n)=\min_{\forall \text{predecessors}}\left[g(\text{predecessor})+d(m,n)w(k)\right] \qquad (27)$$

   - allowed predecessors are determined by the table of local path limitations.
   - the weight $w(k)$ corresponds to the movement from the predecessor to the point $[m,n]$
   - Table contains the equations for computation of partial cummulated distances.

4. The final minimum normalized distance is then given by the last point of grid $\mathbf{g}$:

$$D(\mathbf{O},\mathbf{R})=\frac{1}{N}g(T,R) \qquad (28)$$

Example

d

ref.

4	3	2
2	3	1
4	2	3
0	1	1

test

g

ref.

inf	10	9	7
inf	6	6	5
inf	4	3	5
inf	0	1	2
0	inf	inf	inf

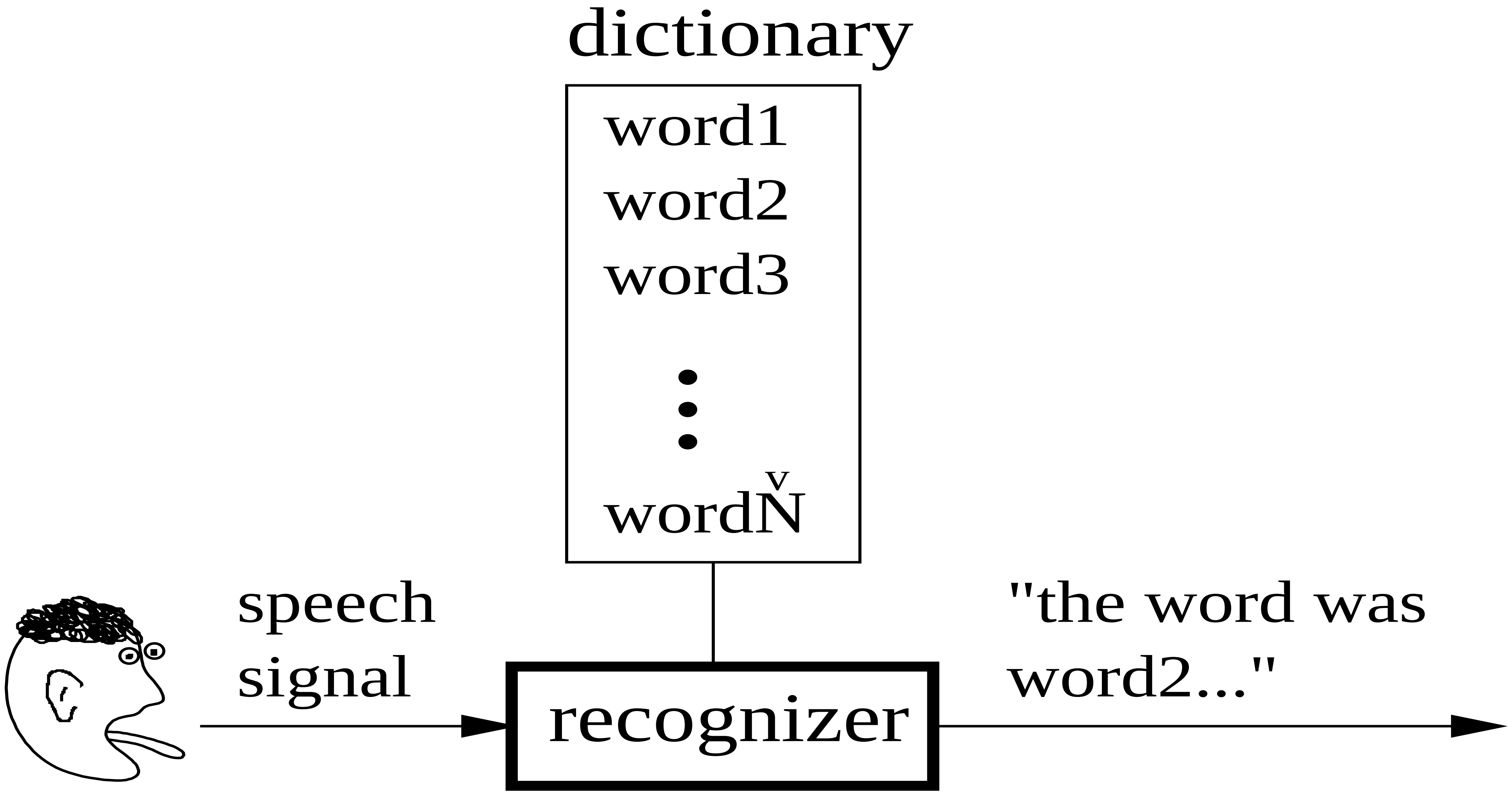
test

Result:

- the DTW distance is: $D = \frac{1}{3+4} 7 = 1$.

- we can back-trace the optimum alignment path (it has 5 steps): $t(k) = [1\ 2\ 2\ 3\ 3]$, $r(k) = [1\ 1\ 2\ 3\ 4]$.

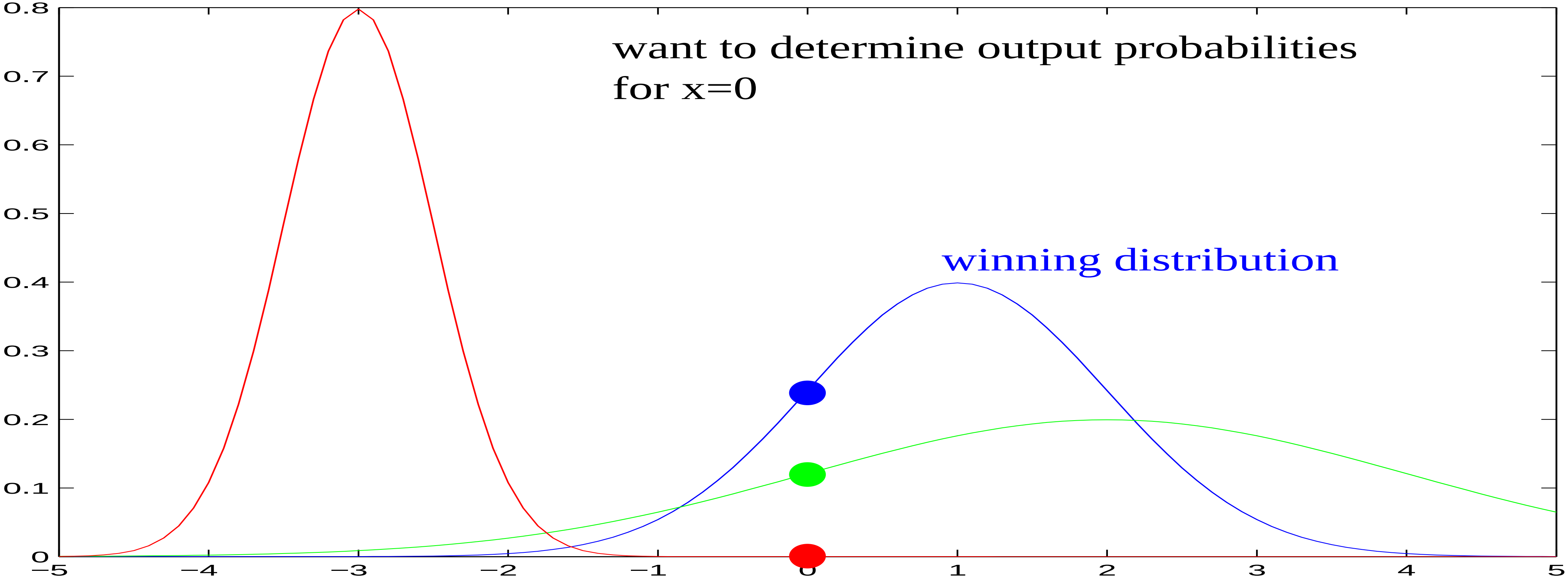
Now it works ;-)


dictionary
word1
word2
word3
wordŇ
speech
signal
recognizer
"the word was
word2..."

BASES OF SPEECH RECOGNITION USING HMM's

Hidden Markov Models - HMM

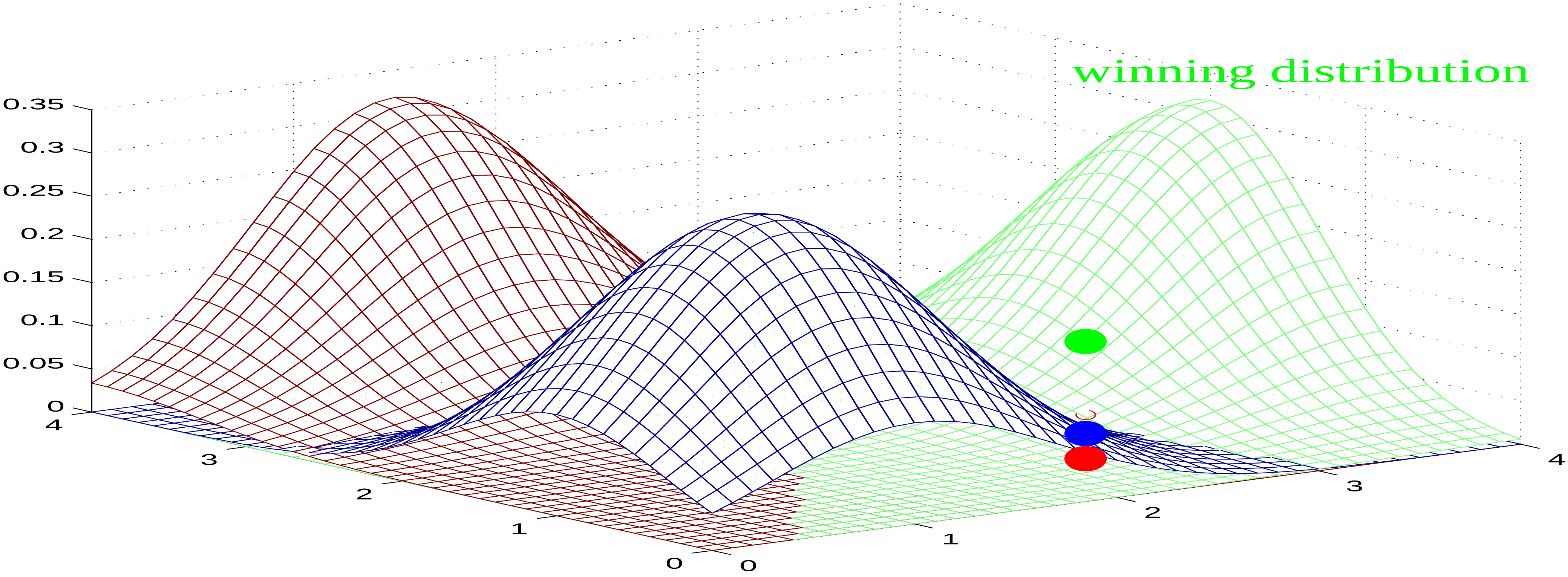
If the test word were represented by one scalar and the reference words by Gaussian distributions...



Remarks

- The value $p(x)$ of the probability density function is **not a probability** (watch out for bloody statisticians ☹). A probability is only $\int_a^b p(x)dx$. But we'll call it **probability**.
- The value $p(x)$ is sometimes called **emission probability** – if the random process could generate vectors, it would generate x with $p(x)$. Our processes won't generate anything, but we'll still call it "emission probability" (to distinguish from transition probas).

. . . but words are **not** represented by scalars but by **vectors** ⇒ multi-dimensional Gaussian distributions.



In reality, the Gaussians have many dimensions (for example 39) — quite bad to imagine/draw, but can always do a projection to 1D, 2D or 3D.

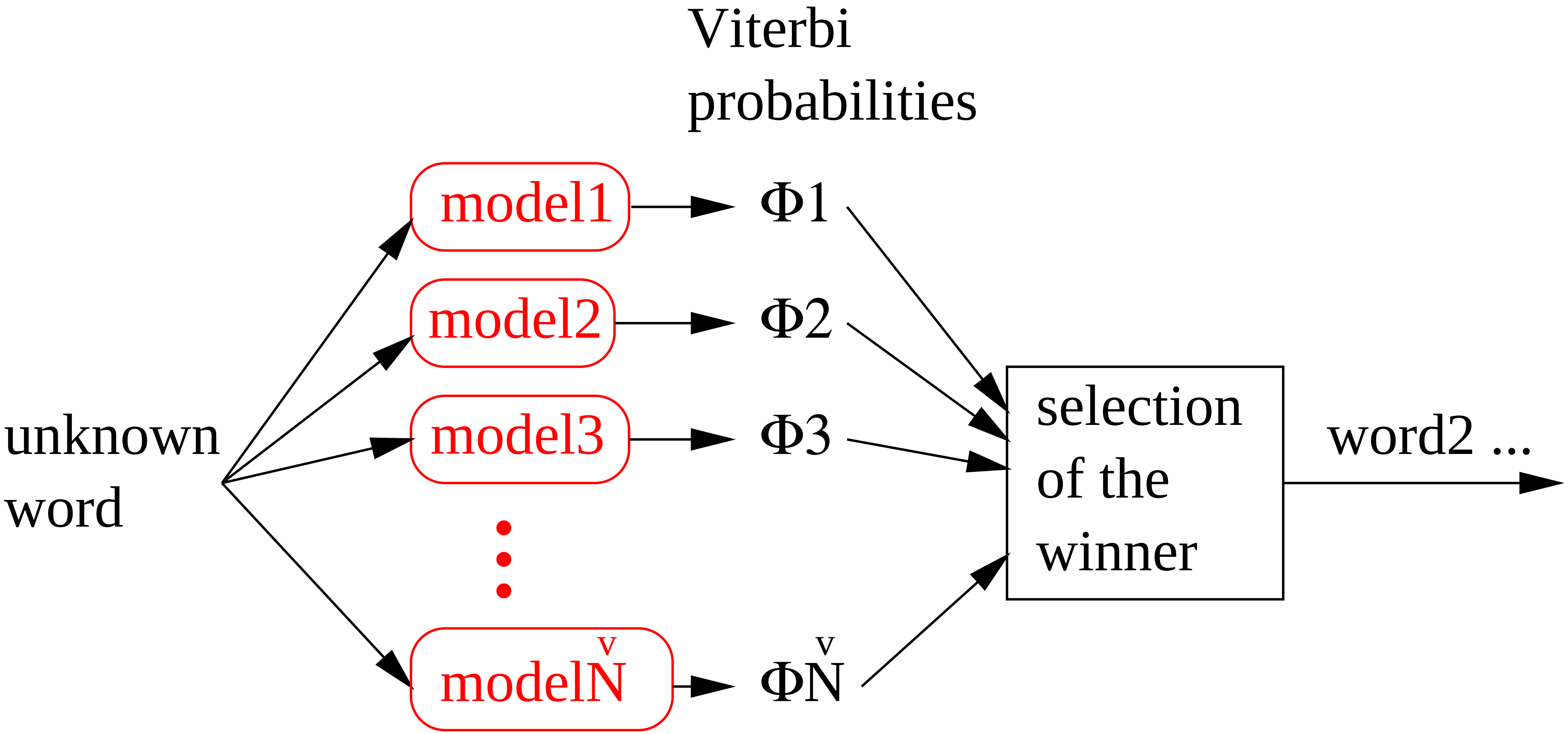
. . . but the words don't have just vector, but many. . .

- Idea 1: have 1 Gaussian per word, process vectors and sum ? $\Rightarrow$ STUPID IDEA ("merde" = "remeède" ???)
- Idea 2: represent each word by a **sequence of more Gaussians**.
  - 1 independent Gaussian per vector ? Hm hm, different nuumbers of vectors (remember DTW) !
  - **a model, where the Gaussians can be repeated !**

These models are called Hidden Markov models HMM

exaplanation on **isolated words**

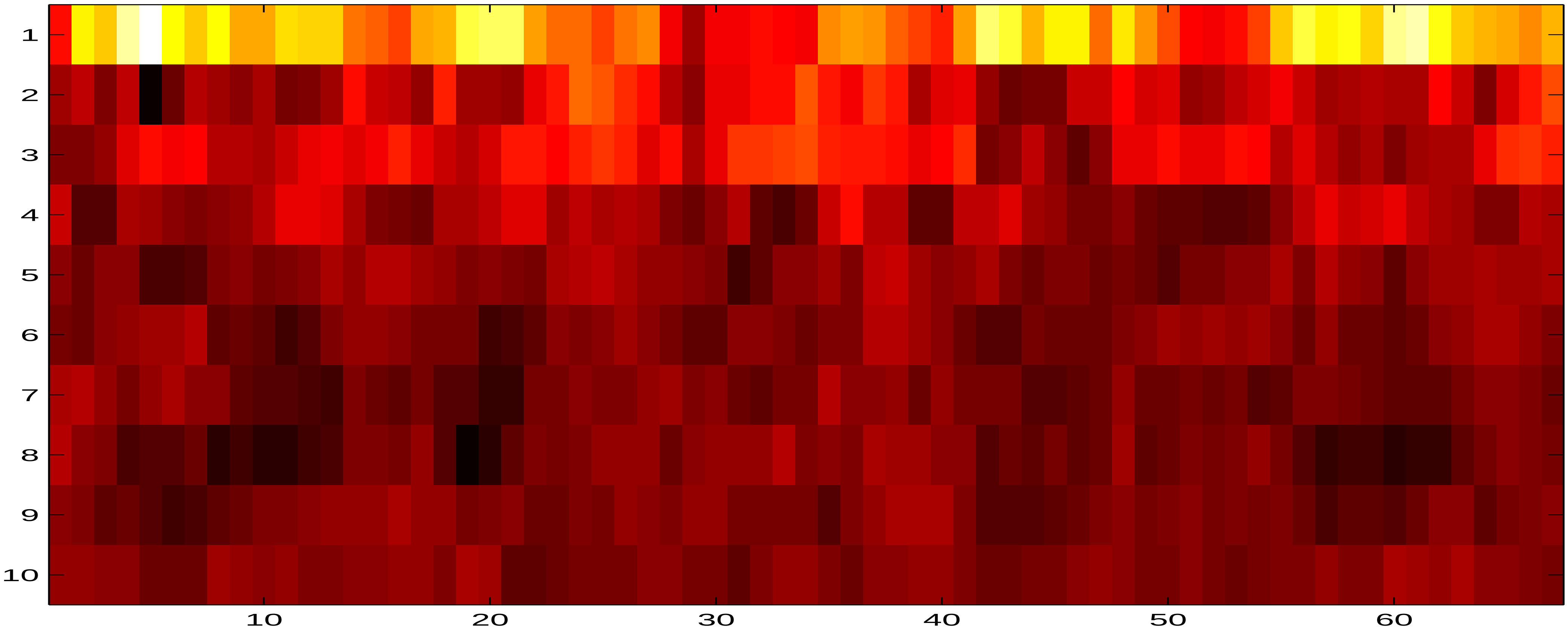
⇒ For each word we want to recognize, we'll need one model:



Now will already need some math. We'll show everything on **one model**, but remember we have to do the same processing for all the models.

Input sequence of vectors:

$$\mathbf{O} = [\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(T)], \tag{29}$$



Probabilistic formulation of the problem

$$i^\star = \arg\max_i \left\{\mathcal{P}(w_i|\mathbf{O})\right\}, \tag{30}$$

where $\mathcal{P}(w_i|\mathbf{O})$ is the conditional probability of the word w_i knowing $\mathbf{O}$.

Illustration of conditional probas:

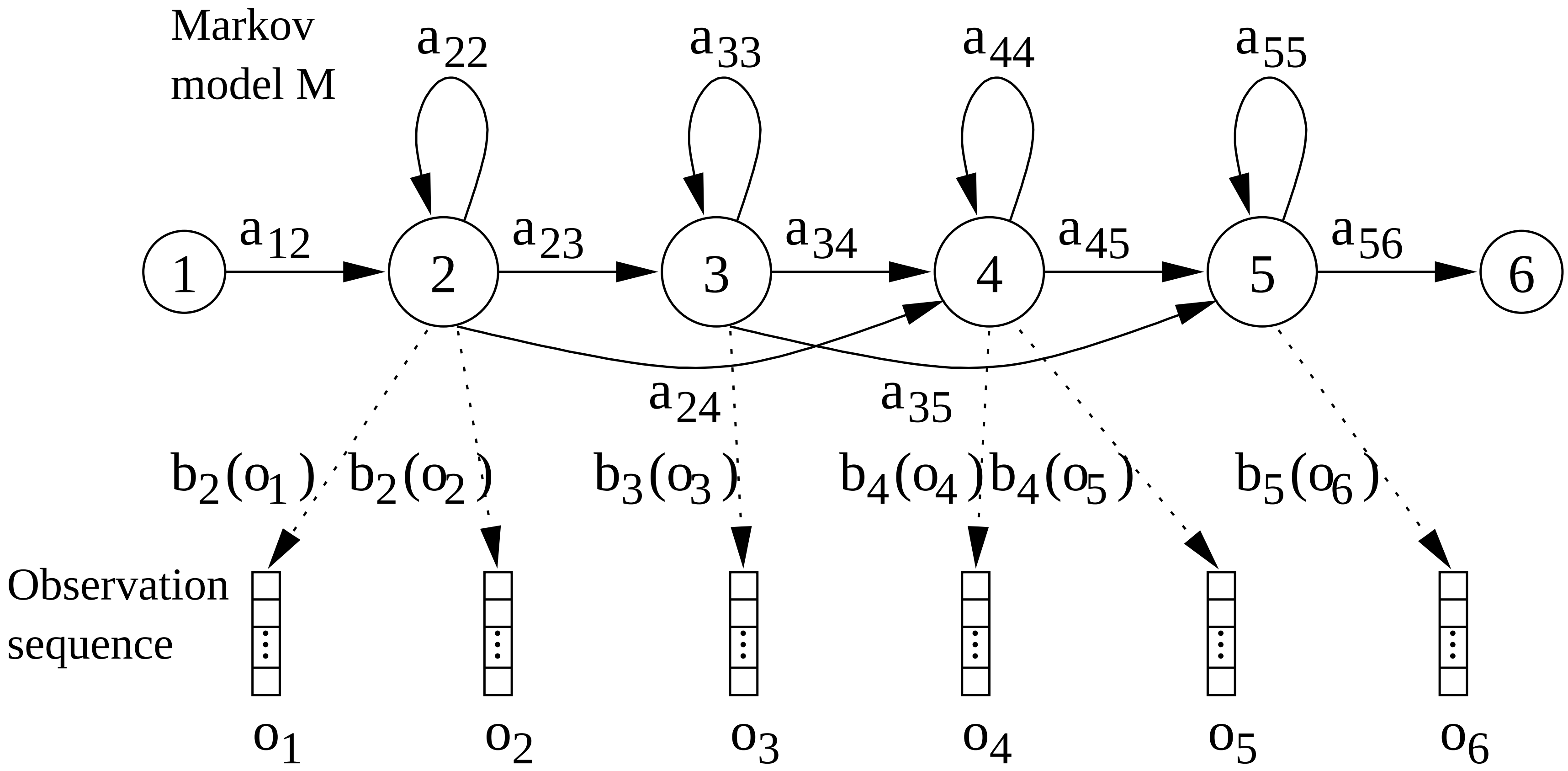
$$\mathcal{P}(problem|\check{S}koda) >> \mathcal{P}(problem|Mercedes). \tag{31}$$

In the speech recognition, we are not able to evaluate $\mathcal{P}(w_i|\mathbf{O})$ directly. The formula of Bayes:

$$\mathcal{P}(w|\mathbf{O}) = \frac{\mathcal{P}(\mathbf{O}|w)\mathcal{P}(w)}{\mathcal{P}(\mathbf{O})}, \tag{32}$$

We are modelling each of words w_i by a model M_i.

Example configuration of an HMM



Transition probabilities a_{ij}

Our example: only three types of transition probas:

- $a_{i,i}$ proba of staying in a state.
- $a_{i,i+1}$ proba of going to the following state.
- $a_{i,i+2}$ proba of skipping the following state.

Matrix of transition probas:

$$\mathbf{A} = \begin{bmatrix} 0 & a_{12} & 0 & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{33}$$

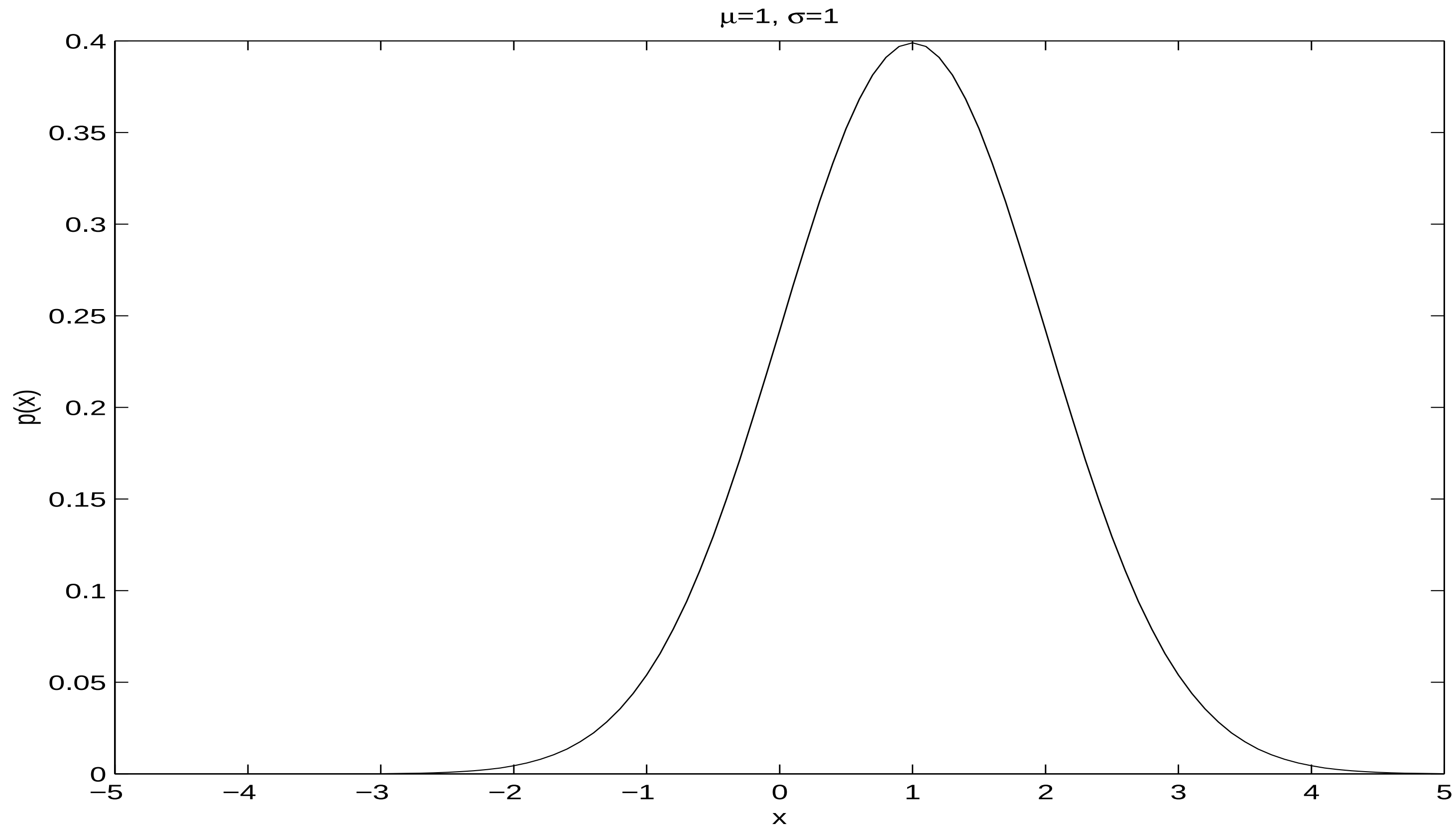
Emission probability density functions (pdfs)

We note the probability of emission of vector $\mathbf{o}(t)$ by i-th state $b_i[\mathbf{o}(t)]$.

Continuous pdfs — continuous density HMMs (CDHMM) Probability density functions given using *distributions* or their sums.

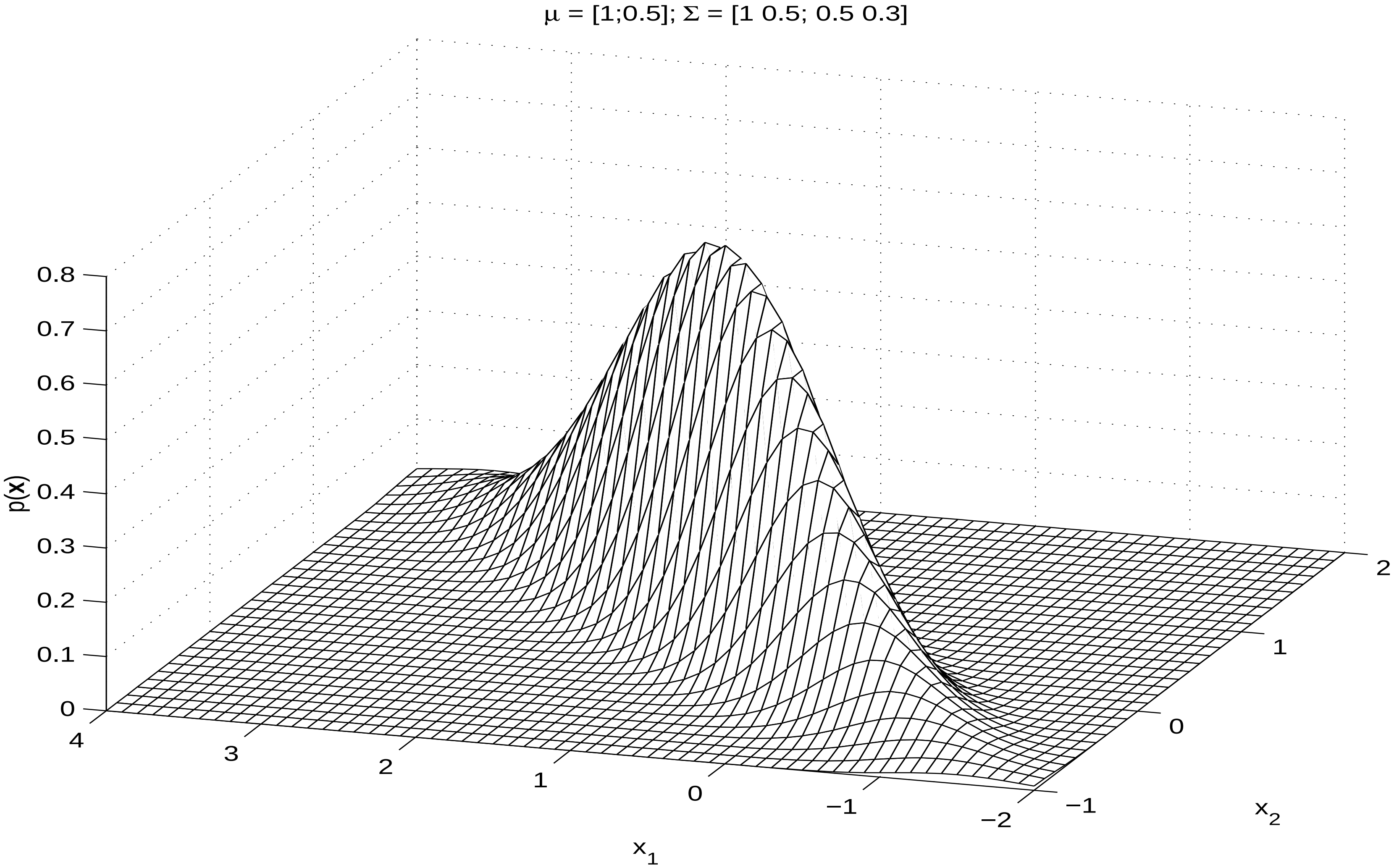
If the vector $\mathbf{o}$ had only one element:

$$b_j[o(t)] = \mathcal{N}(o(t); \mu_j, \sigma_j) = \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{[o(t)-\mu_j]^2}{2\sigma^2}} \tag{34}$$

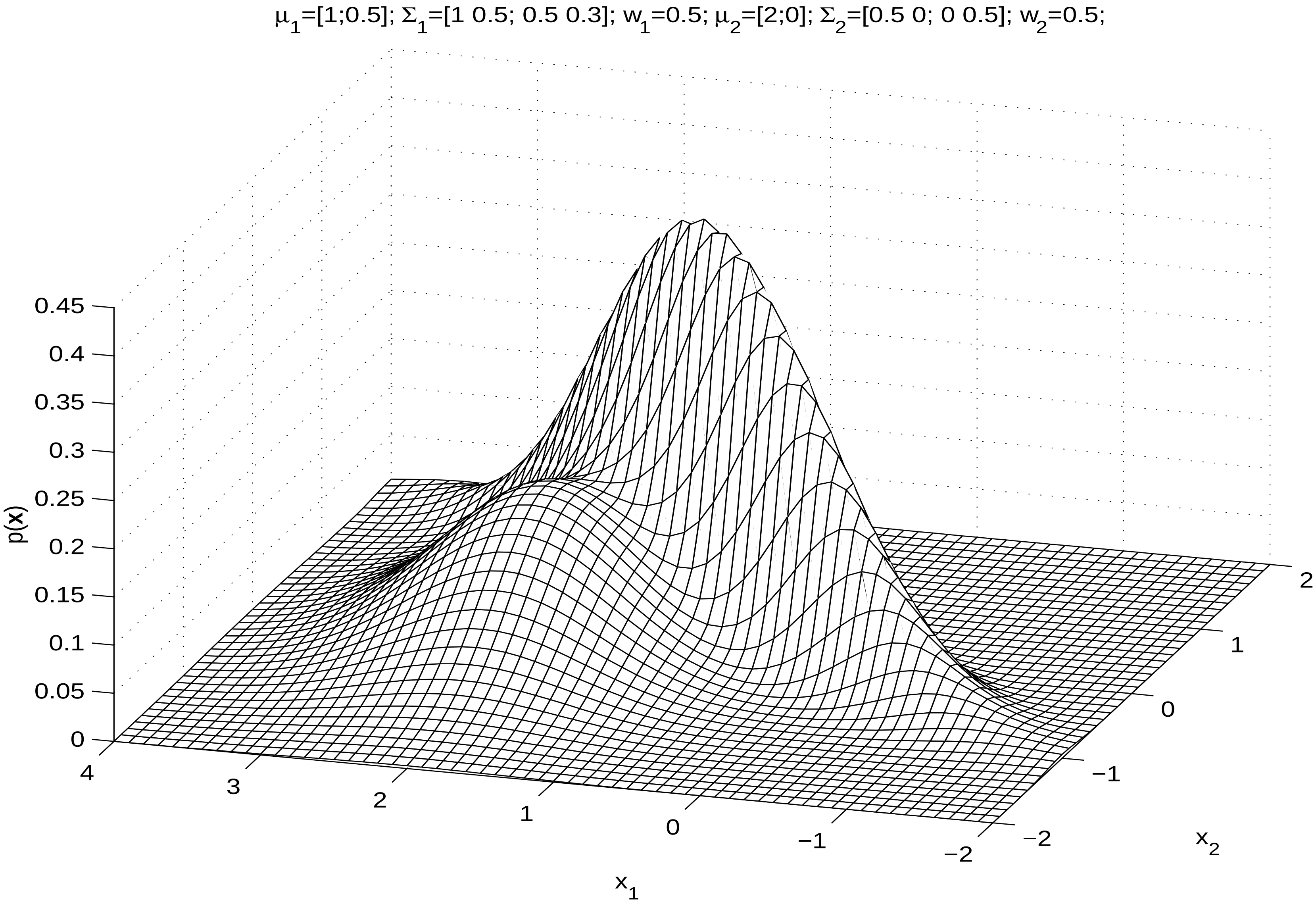

μ=1, σ=1
p(x)
x
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
−5
−4
−3
−2
−1
0
1
2
3
4
5

P-dimensional Gaussian distribution:

$$b_j[\mathbf{o}(t)] = \mathcal{N}(\mathbf{o}(t); \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = \frac{1}{\sqrt{(2\pi)^P |\boldsymbol{\Sigma}_j|}} e^{-\frac{1}{2}(\mathbf{o}(t)-\boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1}(\mathbf{o}(t)-\boldsymbol{\mu}_j)}, \qquad (35)$$



... or even Gaussian mixtures:



Some remarks on distributions

the parameter vectors usually consist of:

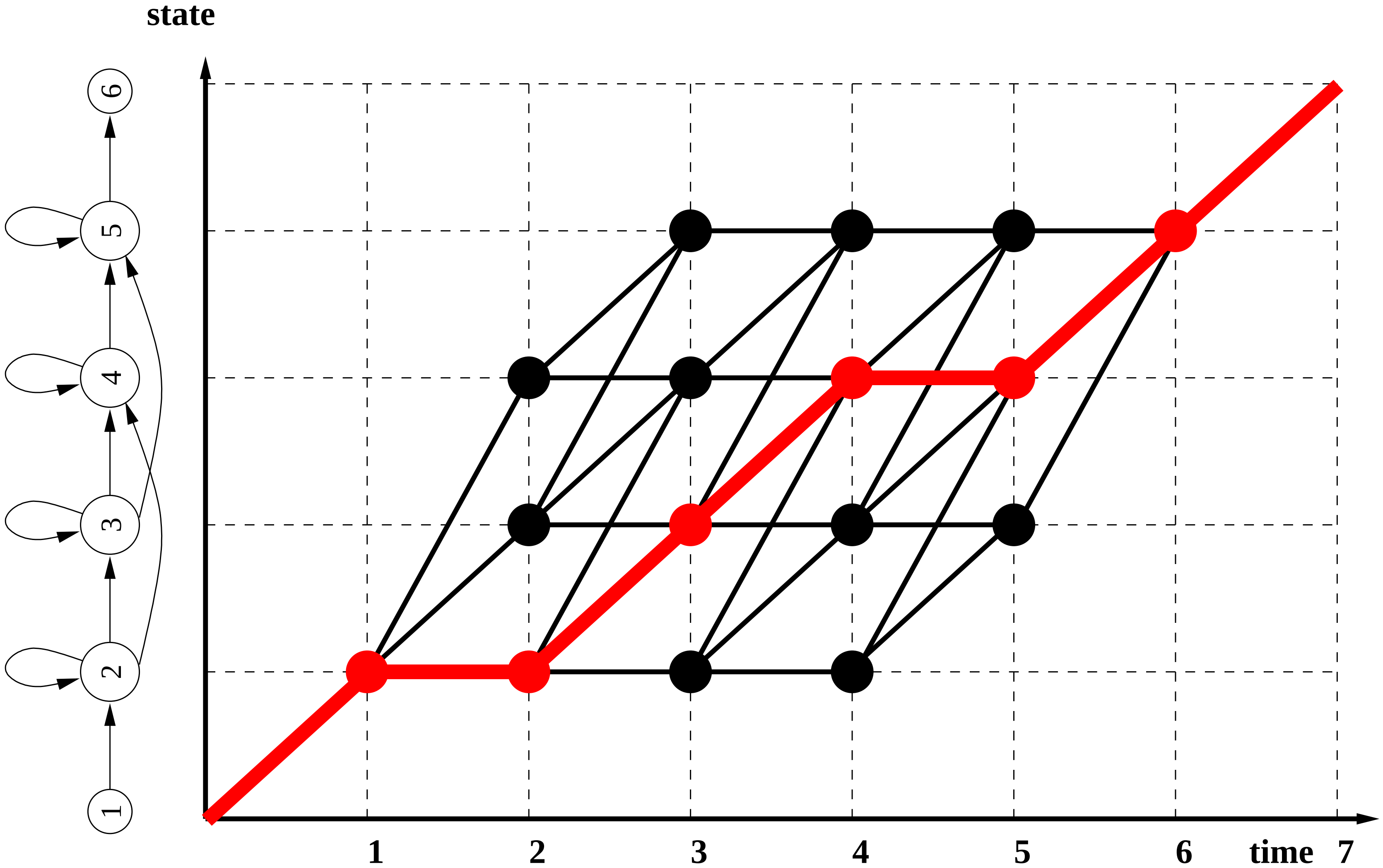
- 12 MFCC (Mel-frequency cepstral) coefficients.
- 12 ΔMFCC coefficients – “velocities”
- 12 $\Delta\Delta$MFCC coefficients – “accelerations”
- E, ΔE and $\Delta\Delta$E – log energy, its velocity and acceleration.

... so that the magic number in speech recognition is 39. This means lots of parameters to train and to store $\Rightarrow$ **Simplifications:**

- if the parameters are not correlated (we assume that the covariance matrix is diagonal) $\Rightarrow$ instead of $P \times P$ covariance coefficients, it is enough to estimate P standard deviations. Consequence: simpler models, the emission pdf is a product of one-dimensional pdfs.
- some sets of parameters can be shared (tied) across states and/or models. Consequence: less parameters, more reliable estimation.

Probability, that the model M generates sequence O

State sequences: states attributed to observation vectors, for example $X = [1\ 2\ 2\ 3\ 4\ 4\ 5\ 6]$.



proba of generation of $\mathbf{O}$ along the path X:

$$\mathcal{P}(\mathbf{O}, X|M) = a_{x(o)x(1)} \prod_{t=1}^{T} b_{x(t)}(\mathbf{o}_t) a_{x(t)x(t+1)}, \tag{36}$$

Two possibilities to define a unique proba, that the model generates the observation sequence:

a) Baum-Welch:

$$\mathcal{P}(\mathbf{O}|M) = \sum_{\{X\}} \mathcal{P}(\mathbf{O}, X|M), \tag{37}$$

where we take the sum of *all possible paths* of length $T+2$ through the model.

b) Viterbi:

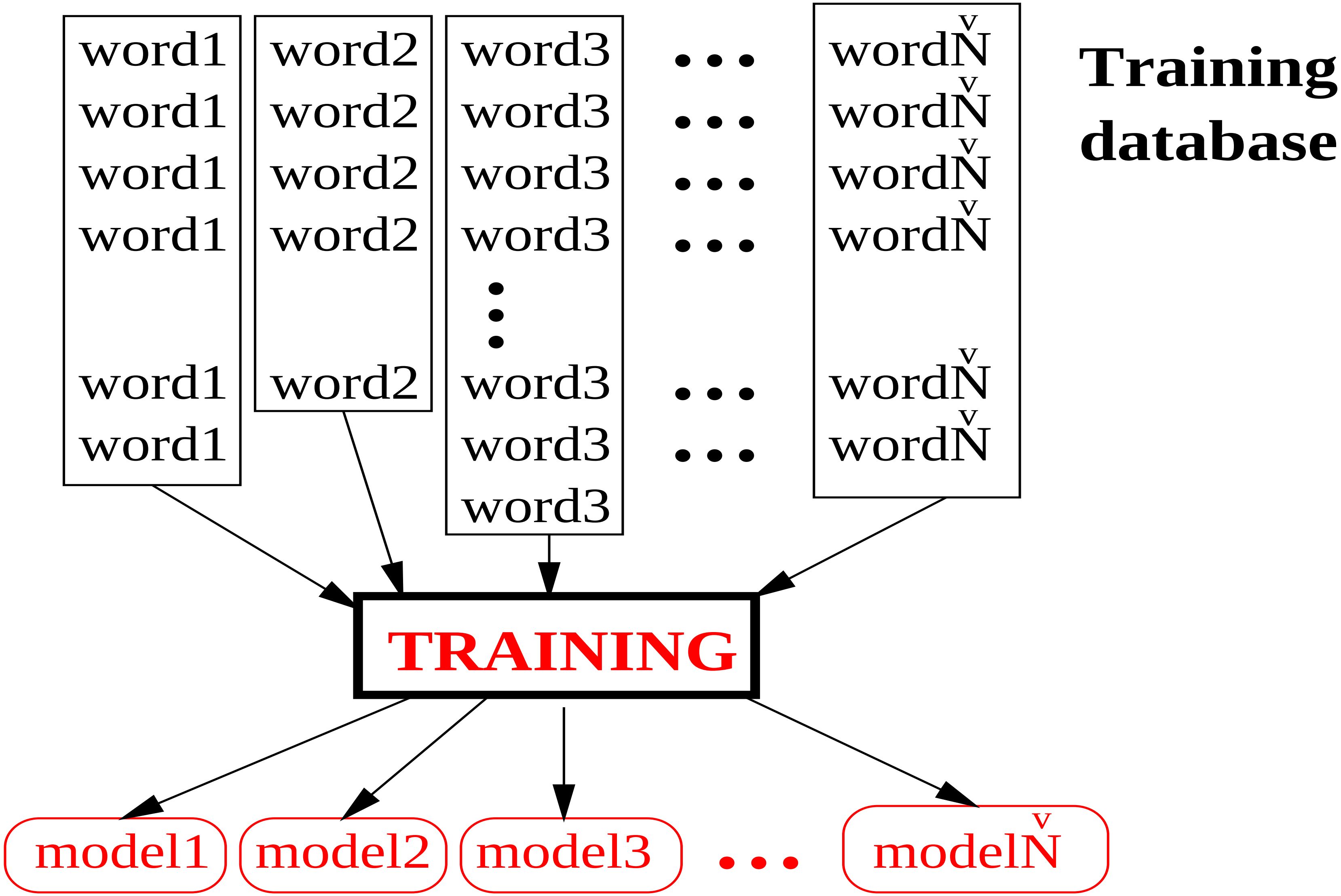
$$\mathcal{P}^{\star}(\mathbf{O}|M) = \max_{\{X\}} \mathcal{P}(\mathbf{O}, X|M), \tag{38}$$

the probability on the optimal path.

Remarks

1. In case of DTW, we have *minimized the distance*. Here, we *maximize the probability*, sometimes also called likelihood and denoted $\mathcal{L}$.
2. Fro the numerical computation of both BAUM-WELCH and VITERBI probas, fast algorithms are known: it is not necessary to evaluate the probas over all possible paths X.

Parameter training (nobody gives us the parameters . . .)



1. the parameters of model are **roughly estimated.**

$$\hat{\boldsymbol{\mu}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{o}(t) \quad \hat{\boldsymbol{\Sigma}} = \frac{1}{T}\sum_{t=1}^{T}(\mathbf{o}(t)-\boldsymbol{\mu})(\mathbf{o}(t)-\boldsymbol{\mu})^T \tag{39}$$

2. vectors are **re-assigned**: "Hard" or "soft" assignment using a state occupation function $L_j(t)$.

3. **estimations are updated**

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} L_j(t)\mathbf{o}(t)}{\sum_{t=1}^{T} L_j(t)} \quad \hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} L_j(t)(\mathbf{o}(t)-\boldsymbol{\mu}_j)(\mathbf{o}(t)-\boldsymbol{\mu}_j)^T}{\sum_{t=1}^{T} L_j(t)}. \tag{40}$$

   ...similar formulas for the computation of transition probas a_{ij}.

Steps 2) and 3) are repeated; stop criterion: given number of iterations, or the probabilities stop to change.

Recognition using the Viterbi decoding

- we should recognize an unknown sequence $\mathbf{O}$.
- we dispose of a dictionary of $\check{N}$ words $w_1 \dots w_{\check{N}}$.
- each of them modeled by a Hidden Markov model: $M_1 \dots M_{\check{N}}$.
- Question is: "Which model would generate $\mathbf{O}$ with the highest probability ?"

$$i^\star = \arg\max_i \{\mathcal{P}(\mathbf{O}|M_i)\} \tag{41}$$

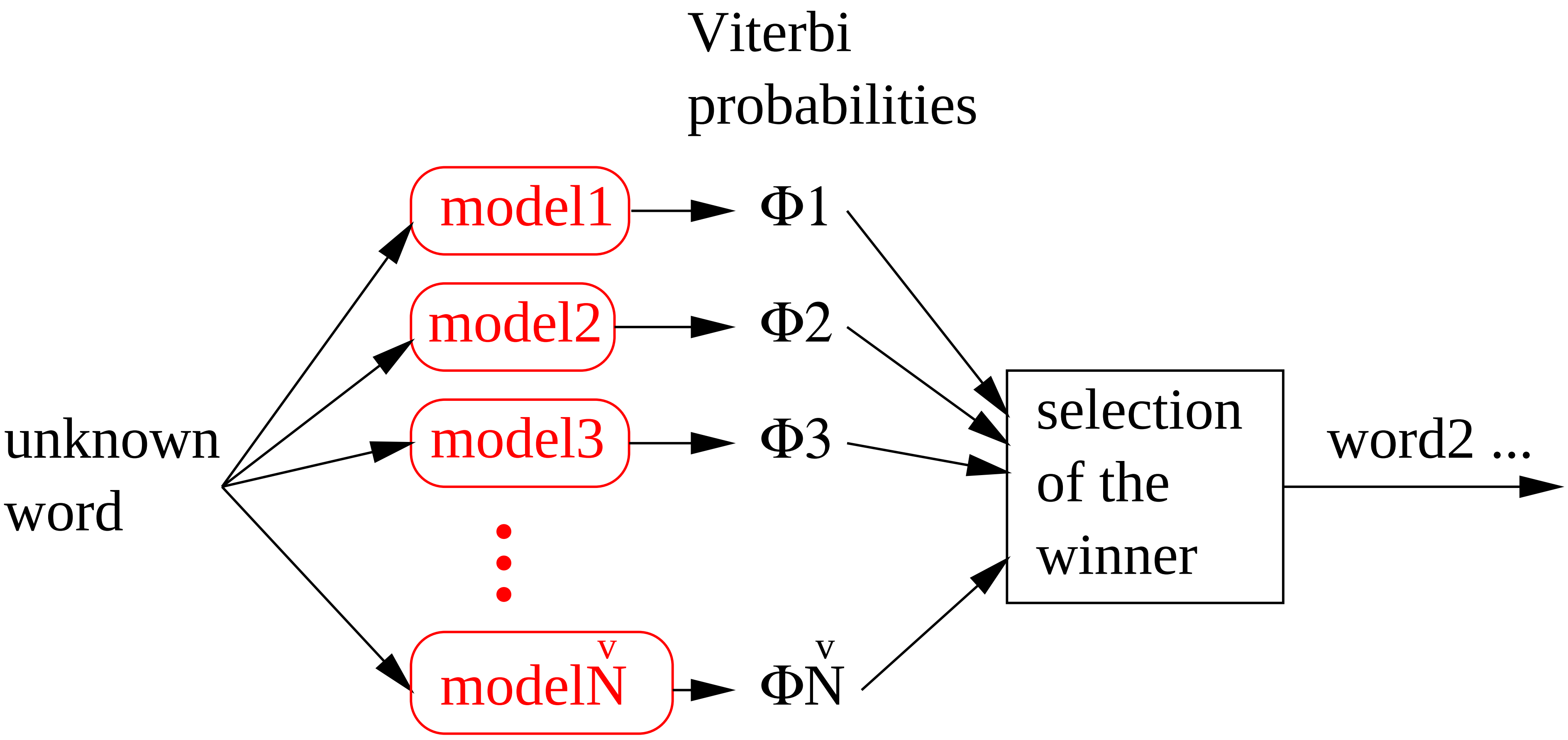
use of Viterbi proba for the most probable sequence of states:

$$\mathcal{P}^\star(\mathbf{O}|M) = \max_{\{X\}} \mathcal{P}(\mathbf{O}, X|M). \tag{42}$$

so that:

$$i^\star = \arg\max_i \{\mathcal{P}^\star(\mathbf{O}|M_i)\}. \tag{43}$$

Isolated word recognition using HMMs



Fast computation of VITERBI probability using beer-jug passing

Each state can hold a *beer-jug*, the level of beer corresponds to log-probability (multiplication = addition in the log domain).

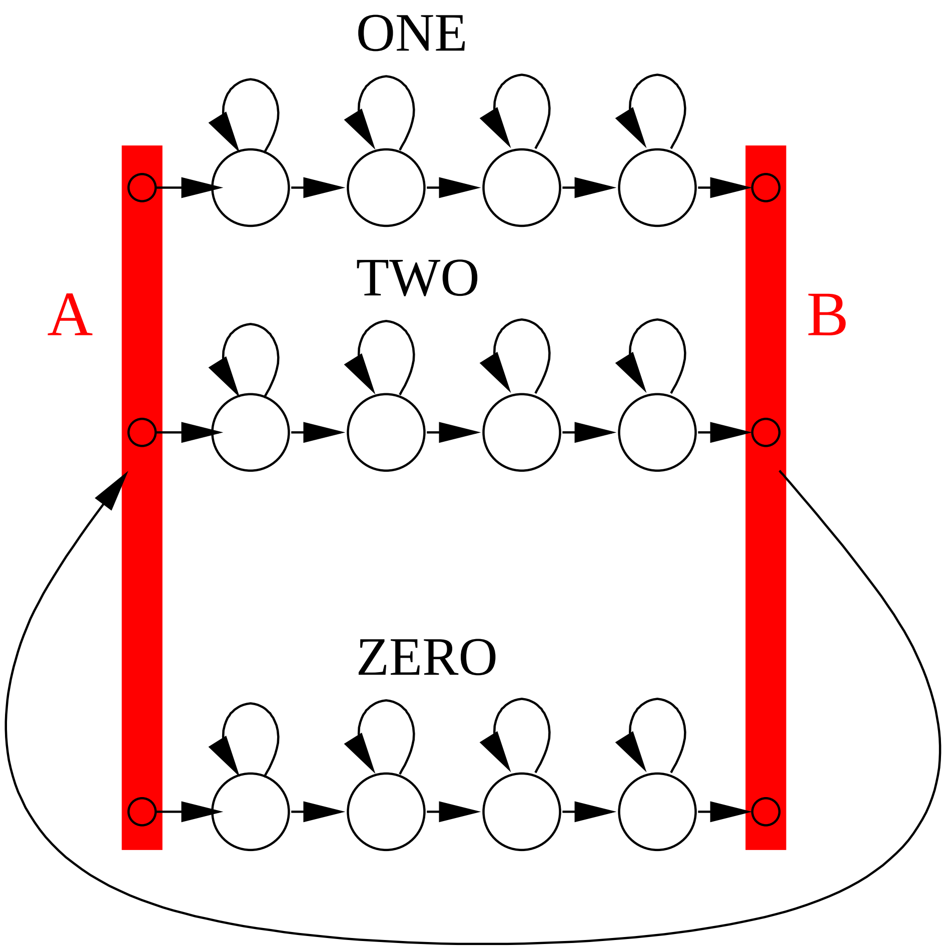
Initialization: insert an empty jug to all initial states of the HMM (normally, this would be just state No. 1).

Iterations: for times $t = 1 \dots T$:

- from each state i containing a beer-jug, send a copy of this jug to all connecting states j, and pour the amount of beer equivalent to $\log a_{ij} + \log b_j[\mathbf{o}(t)]$ in the jug.
- if there are more jugs than one in a state, keep only the fullest one, and discard the others.

Termination: from all states i connected to the output state N, and containing a jug, send a copy of this jug to state N, and poor in the beer equivalent of $\log a_{iN}$. In the final state, select the fullest jug and discarded the others. The amount of beer is equivalent to the log of Viterbi probability: $\log \mathcal{P}^\star(\mathbf{O}|M)$.

Token passing for connected word recognition



- building of a mega-model with all words.
- using initial and final non-emitting states of HMMs to glue the models together (two “tapes”).
- send beer-jugs in the mega-model, there will be one that wins! To allow for strings of digits, we must add a transition from the final “tape” B to the initial state A.

Which words did the best beer-jug cross ?

ISSUES IN SPEECH RECOGNITION...

- features (robust to noise and channel variations).
- language models: is *Jacque Chirak* or *žák Šerák* more probable ?
- speaker normalization/adaptation (supervised good for secretary but bad for word-spotting and terrorists).
- throwing-out garbage.
- how to make ASR multilingual ?
- how to do it without large annotated databases ?
- ...