Image and speech recognition

Lecture notes

2012

Włodzimierz Kasprzak

Project is co-financed by European Union within European Social Fund

CONTENT

Part I – Pattern recognition

1. Pattern recognition 5 - 42
2. Pattern transformation 43 - 77
3. Pattern classification 78 - 114

Part II – Image recognition

4. Image pre-processing 115 - 157
5. Boundary-based image segmentation 158 - 181
6. Region-based image segmentation 182 - 213
7. Model-based object recognition 214 - 231
Part III – Speech recognition

8. Speech signal pre-processing 232 - 250
9. Acoustic feature detection 251 - 274
10. Phonetic speech model 275 - 289
11. Word and sentence recognition 290 - 314

GENERAL REFERENCES

1. Introduction

The GOAL of pattern recognition:

• to analyse (to recognize, to understand) and describe the content of given pattern (digital image, speech signal, etc.).

Example The difference between computer graphics (image synthesis) and image analysis process:

```
<table>
<thead>
<tr>
<th>Description</th>
<th>Graphics</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Analysis</td>
<td>Description</td>
</tr>
</tbody>
</table>
```
**Pattern**

**Patterns** – sets of multi-variable functions that express some physical entity, object, system, etc.

**Domain** of patterns:
\[ \Omega = \{ f_r(x) \mid r = 1, 2, \ldots \} \subset U \]

\[ f_r(x) = \begin{cases} f_{r1}(x_1, x_2, \ldots, x_n) \\ f_{r2}(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_{rm}(x_1, x_2, \ldots, x_n) \end{cases} \]

**Example**

Some patterns
(in a bitmap form)
from the domain of characters:

<table>
<thead>
<tr>
<th></th>
<th>( \Omega_1 )</th>
<th>( \Omega_2 )</th>
<th>( \Omega_3 )</th>
<th>( \Omega_4 )</th>
<th>( \Omega_5 )</th>
<th>( \Omega_6 )</th>
<th>( \Omega_7 )</th>
<th>( \Omega_8 )</th>
<th>( \Omega_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \beta )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \delta )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \zeta )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \iota )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \kappa )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

W. Kasprzak: EIASR 1. Pattern recognition 7

**Pattern recognition approaches**

**Complexity** of patterns \( \leftrightarrow \) pattern recognition approach:

1. **Simple** patterns \( \rightarrow \) **template** matching, **classification**
2. **Complex** patterns (a sequence or set of simple patterns) \( \rightarrow \) pattern-sequence or -structure **model-to-data** matching
3. **Abstract** patterns (e.g. a dynamic scene, continuous speech) \( \rightarrow \) pattern **understanding**.

„2-D image and speech recognition”:
- patterns represented by **speech** or 2-D **image** signals,
- deals with **simple** and **complex pattern** recognition.
Signal, 2-D image

A **signal** is a function of how **one variable** is related to **another variable**, e.g. $a = s(t)$.

**Signal variables:**

- **Amplitude** - the **function value**; may represent voltage, light intensity, sound pressure, etc.
- **Time** – the typical **function parameter** (signal in **time domain**); however, other parameters are used in specific applications.

**2-D Image** – a signal, in which the parameter „time” takes the form of a 2-D space, e.g. $a = I(x, y)$; (signal in **spatial domain**).
**Image scan**

**Scanning**: converting a 2-D image to a 1-D signal.

- **Line scan**
- **Aerial scan**
- **Hilbert scan**

Line scan – not the best one, as local continuity is not preserved.

Aerial („zig-zag”) scan – preserving locality but preferring one direction.

---

**2. 2-D image analysis**

2-D image **analysis systems** contain different **analysis process** at different **data abstraction levels**:

1. **Signal level**: image preprocessing (filtering) - image restoration, image enhancement.

   Example: source separation from mixtures

   - Finite number of mixtures
   - Reconstructed sources
2-D image analysis (2)

2. **Iconic level**: image compression, image registration (normalization), classification of entire image.

Example: image normalization w.r.t. pattern orientation:

(a) Input image, (b) background suppression, (c) foreground border image, (d) rotated (normalized) image

2-D image analysis (3)

3. **Segmentation level**: boundary detection, region detection, texture classification, depth or motion detection.

Example. Detection of characters in license plate regions:

(a) Motion mask, (b), (c) two regions of interest (ROI), d) regions corresponding to characters.
2-D image analysis (4)


Example.

Palm recognition:

(a) 2-D palm model, (b) 3-D palm model, (c) contour detection, (d) discrete feature detection, (e) feature-to-model matching

3. Speech recognition

Typical problems (steps) in a speech recognition system:

1. Audio signal acquisition and speech detection (VAD)
2. Spectral analysis
3. Feature detection

Speech signal
Spectrogram (time-frequency representation)
Sequence of feature vectors
Simple speech classification

Sometimes a simple speech recognition system will do, for example:

- For the control of a device which needs several commands only,
- If selecting a phone number by voice.

**The structure of a simple word recognition system:**

1. Signal acquisition and VAD.
2. Spectrogram analysis with fixed frame number (e.g. 40 frames × 512 samples → 10,240 points).
3. Approximation of the spectrogram image – reducing the image resolution to 640 points (40 x 16).
4. In the learning stage: modelling words from the dictionary by corresponding average low-resolution spectrogram images.
5. Word recognition via direct image classification.
Spectrogram image classifier (2)

Example. Four low-resolution spectrogram images of the spoken words: (a) „koniec” and (b) „OK”.

The coding of spectrogram values: Black - the highest amplitude, Blue - middle range, Yellow - low range, white - near zero range.

Spectrogram image classifier (3)

Example. Spectrograms of the spoken numbers in polish. Significant differences visible.
4. Probability theory

The purpose of applying probability theory in pattern recognition:

- to model the uncertainty of sensor data and of processing results (for example by means of noise),
- the methods of error correction (normalisation, filter) require the statistics of error sources,
- for the judgement of pattern classification and model-based recognition results stochastic distance measures are useful.

Discrete stochastic variable

A non-negative probability density function (pdf) of a discrete stochastic variable $X$: $P(X=x) = p_X(x)$.

Probability of events, specified by an interval of values:

$$P(A \leq X \leq B) = \sum_{A \leq x \leq B} p_X(x)$$

The cumulated density $F_X(x)$ of a stochastic variable $X$:

$$F_X(x) = P(X \leq x) = \sum_{z \leq x} p_X(z)$$

Mean and variance of a pdf:

$$\mu_X = \sum_{x \in \text{dom}(X)} x \cdot p(x) \quad \sigma_X^2 = \sum_{x \in \text{dom}(X)} [(x - \mu_X)^2 \cdot p(x)]$$
Example

The distribution of pixel values (e.g. intensities, colors) in an image can be modelled as a stochastic discrete variable.

The normalized histogram of an image represents relative frequencies of pixel values in given image:

(a) an image, and (b) its intensity histogram

Continuous stochastic variable

The probability value for a given (exact) realization $x$ of a continuous stochastic variable $X$, $P(X=x) = p_X(x)$, is infinitesimally small.

It is meaningful to consider intervals of values:

$$P(A \leq X \leq B) = \int_A^B p_X(x) dx$$

The cumulated density:

$$F_X(x) = \int_{-\infty}^x f(z) dz$$

The pdf of a Gaussian (normally distributed) variable:

$$p_X(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Mean and standard deviation ($\mu$ and $\sigma$) are the only two parameters of this distribution.
Continuous vs. discrete variable

- Stochastic discrete variables represent measurable (observable) entities, i.e. we deal with a finite set of observations – the stochastic features of the observed variable usually change with new observation.
- The probability theory by continuous variables models the hidden stochastic process. Its features may be fixed in time.

**Moments** – the features of a pdf:
- the $k$-th absolute moment of $p$ : $m_k(p) = \int z^k \cdot p(z) \, dz$
- the $k$-th central moment of $p$ :
  $\mu_k(p) = \hat{m}_k(p) = \int (z - m_1(p))^k \cdot p(z) \, dz$

A vector variable

Let $X = [X_1, X_2, \ldots, X_n]^T$ - a vector of stochastic variables.
The mean vector: $E\{X\} = [E\{X_1\}, E\{X_2\}, \ldots, E\{X_n\}]^T$
The covariance matrix:

\[
\Sigma = \begin{pmatrix}
\sigma_{11} & \cdots & \sigma_{1n} \\
\vdots & \ddots & \vdots \\
\sigma_{n1} & \cdots & \sigma_{nn}
\end{pmatrix}
\]

where

$\sigma_{i,j}(X) = E\{(X_i - E\{X_i\})(X_j - E\{X_j\})\}$

**Example.** The $n$-dimensional Gaussian distribution:

$P_X(x) = \frac{1}{\sqrt{2^n \pi^n |\text{det}(\Sigma)|}} e^{-\frac{[x-\mu]^T \Sigma^{-1} [x-\mu]}{2}} \cdot 2^n \pi^n |\text{det}(\Sigma)| |\text{det}(2\pi\Sigma)|$

where “det” - determinant of a matrix.
Bayes rule

**Chain rule** for \( n \) stochastic variables:

\[
p_X([x_1, x_2, \ldots, x_n]) = p_{X_1}(x_1) \cdot p_{X_2|X_1}(x_2 | x_1) \cdot p_{X_3|X_1,X_2}(x_3 | x_1, x_2) \cdot \ldots \cdot p_{X_n|X_1,X_2,\ldots,X_{n-1}}(x_n | x_1, x_2, \ldots, x_{n-1})
\]

For \( n \) stochastic **independent** variables it holds:

\[
p_X([x_1, x_2, \ldots, x_n]) = \prod_{i=1}^{n} p_{X_i}(x_i)
\]

The **Bayes rule** relates conditional probability to joint distribution:

\[
p_{X_i|X_2}(x_1 | x_2) = \frac{p_{X_1,X_2}(x_1, x_2)}{p_{X_2}(x_2)}
\]

Hence:

\[
p_{X_i|X_2}(x_1 | x_2) \cdot p_{X_2}(x_2) = p_{X_2|X_1}(x_2 | x_1) \cdot p_{X_1}(x_1)
\]

Entropy

**Duality of information content and probability:**

- low information of event \( (X=x_i) \): high probability of event.
- high information of event \( (X=x_i) \): low probability of event.

Information content of some realization \( x_i \) of variable \( X \):

\[
I(x_i) = -\log_a P(x_i)
\]

**Entropy** of \( X \) – the mean information content of variable \( X \):

\[
H(X) = -\sum_{x_i \in X} (P(x_i) \cdot \log_a P(x_i))
\]
4. Sampling and digitalization

Signal acquisition and analogue-to-digital (A/D) signal conversion:

- **sampling** (in time or space) and amplitude **digitalization**

**Sampling theorem:** If the sampling frequency of signal \( f(x) \) is at least two times higher than the maximum frequency component, \( \varphi \in (-B_x, B_x) \), of this signal, then the full reconstruction of the original analogue signal from its sample set is possible, \( f_j = f(j \Delta x), j = 0, +/\ - 1, +/\ 2, \ldots; \) where the sampling period is \( \Delta x \leq (1/2B_x) \).

**Duality** of signal representation in the time (space) \( f(t) \) and frequency domain \( F(\omega) \).
Digitalization of amplitude

A sequence of real-valued (analogue) samples: \( \{ f_j \} \).
The corresponding digital-valued samples: \( \{ h_j \} \)
The digitalisation error („noise”): \( n_j = f_j - h_j \)
The **quality of the digitalisation** - the **signal to noise ratio**:

\[
\text{SNR} = 10 \log_{10} \frac{E\{ f_j^2 \}}{E\{ n_j^2 \}}
\]

W. Kasprzak: EIASR 1. Pattern recognition

---

PCM coding

**PCM coding** („pulse code modulation”): optimal digitalization
(amplitude intervals and quantization levels).

Error measure:

\[
\varepsilon = \sum_{i=1}^{L} \frac{a_i}{a_i^4} (f - b_i)^2 p(f) df
\]

\[
\frac{\partial \varepsilon}{\partial b_i} = \int_{a_i}^{a_{i+1}} -2(f - b_i) p(f) df = 0
\]

\[
\frac{\partial \varepsilon}{\partial a_i} = (a_i - b_{i-1})^2 \cdot p(a_i) - (a_i - b_i)^2 \cdot p(a_i) = 0
\]

Solution: intervals \( a_i \) and levels \( b_i \)

\[
a_i = \frac{b_{i-1} + b_i}{2}, \quad i = 2,3,...,L = 2^B
\]

\[
b_i = \frac{\int_{a_{i-1}}^{a_i} f \cdot p(f) df}{\int_{a_i}^{a_{i+1}} p(f) df}, \quad i = 1,2,...,L
\]
5. Optimization theory elements

LSE (least square error)

2D case: to establish a linear relation between two quantities, $y$ and $x$: $y = ax + b$ or $ax + b - y = 0$,
given $N$ observations:

$ax_1 + b - y_1 = \varepsilon_1$

$ax_2 + b - y_2 = \varepsilon_2$

$\ldots \ldots \ldots$

$ax_N + b - y_N = \varepsilon_N$.

Goal function: $U(a, b) = \varepsilon_1^2 + \varepsilon_2^2 + \ldots + \varepsilon_N^2 = \sum \varepsilon_i^2 = |\varepsilon|^2$

Minimization of $U(a, b)$: $\frac{\partial U}{\partial a} = 0, \frac{\partial U}{\partial b} = 0$.

Solution:

$$
\begin{bmatrix}
    a \\
    b
\end{bmatrix} = \frac{1}{N \sum x_i^2 - (\sum x_i)^2} \begin{bmatrix}
    N & -\sum x_i \\
    -\sum x_i & \sum x_i^2
\end{bmatrix} \begin{bmatrix}
    \sum (x_i, y_i)
\end{bmatrix}
$$

Moore-Penrose pseudo-inverse

The Moore-Penrose pseudo-inverse is a general solution to the system of $m$ linear equations with $n$ unknowns:

$b = Ay$, $b \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$

The solution is:

$y = A^\dagger b$

where matrix $A^\dagger$ is the Moore-Penrose pseudo-inverse.

When $A$ is full rank $A^\dagger$ can be directly calculated:

- case $m = n$ : $A^\dagger = A^{-1}$
- case $m > n$ : it is the general linear LS solution, which minimizes the squared error (second-order norm) $|b - Ay|^2$

$A^\dagger = (A^T A)^{-1} A^T$

- case $m < n$ : (it minimizes the norm $|y|^2$) $A^\dagger = A^T (A A^T)^{-1}$
Nonlinear LSE (1)

Assume, there is some number $N$ of data samples: $(x_i, y_i) \in \mathbb{R}^2$

The relation between variables $x$ and $y$ is approximated by a parametric non-linear function: $y = f(x \mid p)$, where $p \in \mathbb{R}^n$, $N \geq n$, is the parameter vector.

**Goal:** to find the minimum of the **goal** (objective) function:

$$\Phi(p) = \frac{1}{2} r^2(p) = \frac{1}{2} \sum_{i=1}^{N} r_i^2(p) = \frac{1}{2} \sum_{i=1}^{N} [y_i - f(x_i \mid p)]^2$$

Look at the **gradient** of the goal function:

$$\nabla \Phi(p) = \sum_{i=1}^{N} r_i(p) \nabla r_i(p) = J(p)^T r(p)$$

The **Jacobi** matrix: $$\{J(p)\}_{ij} = \frac{\partial r_i(p)}{\partial p_j}$$

Nonlinear LSE (2)

Second-order derivatives, are in the **Hessian**:

$$H(p) = \nabla^2 \Phi(p) = J(p)^T J(p) + \sum_{i=1}^{N} r_i(p) \nabla^2 r_i(p)$$

for small errors $r$, approximated as:

$$\nabla^2 \Phi(p) = J(p)^T J(p)$$

**Solutions of the nonlinear LSE problem**

1) **Gradient descent search** – an iterative first-order update rule:

$$p_{i+1} = p_i - \lambda \nabla \Phi(p_i)$$

2) **Gauss-Newton search** - second-order based update rule:

$$p_{i+1} = p_i - (\nabla^2 \Phi(p_i))^{-1} \nabla \Phi(p_i)$$

3) **Levenberg-Marquardt** algorithm – uses a combined rule:

$$p_{i+1} = p_i - (H(p_i) + \lambda \text{diag}(H(p_i)))^{-1} \nabla \Phi(p_i)$$
Linear programming

**Linear programming (LP), or linear optimization:**
- optimization of a *linear* objective function,
- subject to *linear* equality and *linear* inequality constraints.

The solution is within a **convex polyhedron**, a set defined as the intersection of finitely many half spaces.

Linear programming problems in **canonical form**:
- Maximize the objective function \( \hat{x} = \arg \max_x c^T x \)
- Subject to constraints \( Ax \leq b \), where \( x \geq 0 \).

**LP in MATLAB (Optimization Toolbox):** BINTPROG, LINPROG

Quadratic programming

In **Quadratic Programming (QP)** the goal (objective) function is **quadratic** and constraints are **linear**:

\[ \hat{x} = \arg \min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} x^T H x + t^T x \]

subject to
\[ a_i^T x = b_i , \quad i = 1, ..., m \]
\[ c_i^T x \geq d_i , \quad i = 1, ..., p \]

where \( H \) is a strictly positive-definite matrix of dimension \( n \times n \),
\( t \) – a vector of length \( n \), \( a_i, c_i \) – vectors of length \( n \), and \( b_i, d_i \) – scalars.

In MATLAB - the **quadprog** routine.
Quadratic programming (2)

Simple: only equality constraints appear in the QP problem: \( A \cdot x = b \).

If the number of equality constraints is the same as the number of independent variables \((m = n)\), constraints uniquely determine the solution to the QP problem: \( x = A^{-1} \cdot b \).

If the number of equality constraints is smaller than the number of independent variables \((m < n)\), use the elimination method for QP.

Solution of the QP. E.g. the active set method:

- In consecutive iterations of such algorithm QP tasks with equality constraints are solved, i.e. only active constraints in current solution point are taken into account.
- The solution to current equality QP gives the direction in which the point is changed. The increment is a minimum value that changes one previously inactive constraint into an active one.

6. Estimation theory

Estimation of a stochastic system

The observation \( z(t) \) of the system state \( s \) is distorted by noise \( w(t) \):
\[
    z(t) = h(s, w(t)) , \quad t = 1, 2, 3, .., N
\]

An estimation of state \( s \) is needed, as a direct solution is not available. If the added noise is of zero-mean then we have a bias-free estimator.

Stochastic variable estimator (\( s \) is a stochastic variable):
1. The ML estimator (without prior information).
2. The MAP estimator MAP (explores the prior information).

Deterministic variable estimator:
1. The LSE estimator.
2. The MMSE estimator.
ML, MAP

The **ML (maximum likelihood) estimator**

The conditional observation distribution is available

\[ p(z_1, z_2, \ldots, z_n \mid s) \]

The ML estimate:

\[ s^{ML}(t) = \arg \max_s p(z_1, z_2, \ldots, z_n \mid s) \]

The **MAP (maximum posterior) estimator**

A non-uniform pdf \( p(s) \) is available.

The MAP estimate - to maximise a weighted combination of observations and prior \( p(s) \):

\[ s^{MAP}(t) = \arg \max_s p(z_1, z_2, \ldots, z_n \mid s) p(s) \]

When \( p(s) \) is uniform (no prior information, flat distribution function) then MAP simplifies to ML.

W. Kasprzak: EIASR 1. Pattern recognition

LSE, MMSE

The **LSE (least square error) estimate**:

\[ s^{LS}(t) = \arg \min_s \sum_{i=1}^{t} [z(i) - h(i, s)]^2 \]

For a Gaussian noise with zero mean value the LSE estimate is equivalent to the ML estimate.

The **MMSE (minimum mean square error) estimator**

The MMSE estimate is the expected value of state w.r.t. given measurements:

\[ s^{MMSE} = \arg \min_{\hat{s}} E[(\hat{s} - s)^2 \mid z(1), z(2), \ldots, z(t)] = \]

\[ = \arg \min_{\hat{s}} \sum_{i=1}^{t} [z(i) - h(\hat{s})]^2 \cdot p(z(i)) \]

For a non-stochastic variable \( z \) the MMSE estimate is equivalent to the LSE and its value is the current real value of the state variable.
1. Classifier design

A simple pattern classification problem: to represent the pattern by a numeric feature vector, \( c \in \mathbb{R}^d \), and to assign to it an appropriate discrete class label \( \kappa \):

\[
\zeta(c) = \begin{cases} 
\mathbb{R}^d & \rightarrow \{1, 2, \ldots, K\} \\
c & \rightarrow \kappa 
\end{cases}
\]

The design of a simple pattern classifier depends on:

1. the quality of the feature set that is available for training (learning) – e.g. maximise the concentration of features from the same class and/or maximise the separation between features from different classes,

2. the expected classification error – due to classifier type
Feature space

Space transforms, like *Fast Fourier Transform* or *Discrete Cosine Transform*, are orthogonal and fixed transformations - set independently of the learning set. We review basic linear transforms of the pattern space, like PCA, LDA, ICA, that depend on the data training set and which optimize some criteria set on this data.

The naive basis of pattern space:

*Is there another basis, which is a linear combination of the original basis, that best expresses our data?*

\[
B = \begin{bmatrix}
    b_1 \\
    b_2 \\
    \vdots \\
    b_m \\
\end{bmatrix} = \begin{bmatrix}
    1 & 0 & \cdots & 0 \\
    0 & 1 & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & 1 \\
\end{bmatrix} = I
\]

Space transformation criteria

Let: \( K \) - number of classes; \( N \) - number of training samples (vectors of size \( n \)), \( jf_k \) – a training sample of the \( k \)-th class; \( jc_k \) – feature for the \( j \)-th training sample of the \( k \)-th class. We are seeking linear transformations of the pattern space:

\[
jc = \Phi \cdot jf,
\]

that are optimal with respect to some goal functions (criteria).

**Goal functions:**

1) Average squared distance between two features (→ **PCA – principal component analysis**):

\[
s_1 = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (jc - jc)^T (jc - jc)
\]
Transformation criteria $s_2$ and $s_3$

2) Average squared distance between features from two different classes:

$$s_2 = \frac{2}{K(K-1)} \sum_{k=1}^{K} \sum_{l=1}^{K} \frac{1}{N_k N_l} \sum_{i=1}^{N_k} \sum_{j=1}^{N_l} (c_{k-l} c_l)^T (c_{k-l} c_l)$$

3) Average squared distance between two features from the same class:

$$s_3 = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{N_k} \sum_{i=1}^{N_k} \sum_{j=1}^{N_k} (c_{k-l} c_k)^T (c_{k-l} c_k)$$

A popular form is the combination of $s_2$ and $s_3$, which gives a criterion, called Fisher’s information, applied in linear discriminate analysis - LDA).

The optimal transformation matrix w.r.t. $s_1$ - $\Phi^{(l)}$ contains eigenvectors $\varphi_v^{(l)}$ of some symmetric matrix $Q^{(l)}$. It holds:

$$Q^{(l)} \varphi_v^{(l)} = \lambda_v^{(l)} \varphi_v^{(l)} ,$$

where $\lambda_v^{(l)}$, $v = 1, ..., n$, are eigenvalues of $Q^{(l)}$.

- To maximize the criteria $s_1$ or $s_2$ : compute $n$ eigenvectors $\varphi_v^{(l)}$ or $\varphi_v^{(2)}$ related to $n$ largest eigenvalues $\lambda_v^{(l)}$ or $\lambda_v^{(2)}$, $v = 1, ..., n$; of the matrix $Q^{(l)}$ or $Q^{(2)}$, respectively.
- To minimize $s_3$ : compute the eigenvectors related to the smallest eigenvalues of some matrix $Q^{(3)}$. 

$$\Phi^{(l)} = \begin{bmatrix} (\varphi_1^{(l)})^T \\ (\varphi_2^{(l)})^T \\ \vdots \\ (\varphi_n^{(l)})^T \end{bmatrix}$$
Data covariance matrices

Symmetric matrices, \( Q^{(1)} \) (\( l=1,2,3 \)), required by the criteria \( s_l \):

\[
Q^{(1)} = R - mm^T,
\]

\[
R = \frac{1}{N} \sum_{j=1}^{N} jf \cdot jf^T, \quad m = \frac{1}{N} \sum_{j=1}^{N} jf, \quad jf \in \omega
\]

\[
Q^{(2)} = \frac{1}{K} \sum_{k=1}^{K} R_k - \frac{1}{K(K-1)} \sum_{k=2}^{K} \sum_{l=1}^{k-1} (m_k m_l^T + m_l m_k^T),
\]

\[
R_k = \frac{1}{N_k} \sum_{j=1}^{N_k} jf_k \cdot jf_k^T, \quad m_k = \frac{1}{N_k} \sum_{j=1}^{N_k} jf_k, \quad jf_k \in \omega_k
\]

\[
Q^{(3)} = \frac{1}{K} \sum_{k=1}^{K} (R_k - m_k m_k^T)
\]

Proof idea

1) Show that every criterion \( s_l \) can be expressed as:

\[
s_l = \sum_i x_i^T Q x_i
\]

2) The sum of positive values attains its maximum if all its components attain their maxima.

3) Use repeatedly the Hotelling thesis:

„Let \( Q \) be some symmetric and positively-defined matrix and \( x \) – a vector of appropriate size.

Define: \( a = x^T Q x \).

Then \( a \) is of maximum possible value if \( x \) is an eigenvector of matrix \( Q \) related to the largest eigenvalue of \( Q \).”
Conclusion

• The criteria are expressed as sums of eigenvalues of appropriately defined data covariance matrices:

\[ s_l = \sum_{v=1}^{n} \varphi_v^T Q^{(l)} \varphi_v = \sum_{v=1}^{n} \lambda_v^{(l)} \]

• If some number, \( N \geq n \), of linearly independent vector samples are available \( \{ jf | j = 1, 2, \ldots, N \} \) (every one is of size \( n \)) then the matrix \( Q \) of size \( n \times n \) is of rank \( n \) and there exist \( n \) different, positive- and real-valued eigenvalues \( (\lambda_1 > \lambda_2 > \ldots > \lambda_n) \) and \( n \) corresponding eigenvectors.

2. PCA

The transformation optimizing \( s_1 \) is Principal Component Analysis (also called Karhunen-Loeve transform):

1) Get the zero-mean covariance matrix \( Q^{(1)} \) of samples \( \{ jx \} \).
2) Obtain its eigenvectors by the eigenvector decomposition or SVD algorithm:

\[ Q^{(1)} = \sum_{r=1}^{n} \lambda_r \varphi_r \varphi_r^T = \Lambda U U^T \]

where \( U \) represents the matrix of eigenvectors \( \varphi_r \) of \( Q^{(1)} \) and \( \Lambda \) is the diagonal matrix of positive eigenvalues \( \lambda_r \), while \( n \) is the rank of the matrix \( Q^{(1)} \).

3) Select the feature space dimension, \( m \leq n \). The feature vector \( jc_m \) corresponding to sample \( jx \) is: \( jc_m = \Lambda_m U_m^T jx \) (non-normalized feature) or \( jc_m = \Lambda_m^{-1/2} U_m^T jx \) (normalized).
PCA (2)

**PCA**: a rotation of an orthogonal coordinate system that orders the axis to capture largest variances of data samples. If \( n \) is the input vector size then some \( m < n \) principal components represent interesting structure, while those with lower variances represent noise.

![Diagram of PCA transformation](image)

Inverse-PCA transform

Let: \( \text{dimension}(\mathbf{c}_m) = m \); \( \text{dimension}(\mathbf{x}) = n \).

In PCA usually, \( m < n \), even most often, \( m << n \).

Hence feature extraction by PCA is **pattern encoding** or **compression**.

**Inverse PCA (decoding, decompression)**

Use “non-normalized” feature coefficients: \( \mathbf{\tilde{c}}_r = \lambda_r \mathbf{\varphi}_r^T \mathbf{x} \)

As the matrix \( \mathbf{U} \) (in PCA) is an orthogonal matrix, its inverse is a transpose. Thus the inverse transformation to PCA is:

\[
\hat{\mathbf{x}} = \mathbf{U} \mathbf{\tilde{c}} = \sum_{r=1}^{m} \mathbf{\tilde{c}}_r \cdot \mathbf{\varphi}_r
\]

and \( \mathbf{x} \) is reconstructed by a linear combination of eigenvectors and feature vector coefficients.
Example

Illustration of PCA-based image compression-decompression. Blocks of size 8x8 are compressed (encoded) and decompressed (decoded) using 16 principal components (thus n=64, m=16).

Example (2)

Results of block-based compression-decompression by PCA and inverse PCA with 1-5 principal components.
3) Linear discriminate analysis

LDA determines an optimal separation (in the sense of \( s_2 \) and \( s_3 \)) of features for samples from different classes.

**The two-classes case**

Let \( N \) samples, \( \{x_1, ..., x_N\} \), are given in \( \mathbb{R}^n \), from two classes (\( \Omega_1, \Omega_2 \)). We look for a linear transformation:

\[
\mathbb{R}^n \rightarrow \mathbb{R} : y = w^T x
\]

The mass centre of region \( \Omega_i \) (\( i = 1,2 \)):

\[
m_i = \frac{1}{N_i} \sum_{x \in \Omega_i} x, \quad (i = 1,2)
\]

In the output (transformed) space:

\[
\tilde{m}_i = \frac{1}{N_i} \sum_{y \in Y_i} y = \frac{1}{N_i} \sum_{x \in \Omega_i} w^T x = w^T m_i, \quad (i = 1,2)
\]

\[
| \tilde{m}_1 - \tilde{m}_2 | = | w^T (m_1 - m_2) |
\]

---

**Two-classes LDA**

The transformed within-class variability: \( \tilde{s}_i^2 = \sum_{y \in Y_i} (y - \tilde{m}_i)^2. \)

Total within-class variability: \( (\tilde{s}_1^2 + \tilde{s}_2^2) \)

**Definition**

The linear Fisher’s discriminate function for 2 classes is \( y = w_o^T x \), where \( w_o = \text{arg max} \ J(w) \)

and \( J(w) \) is the goal function (called Fisher’s information):

\[
J(w) = \frac{| \tilde{m}_1 - \tilde{m}_2 |^2}{\tilde{s}_1^2 + \tilde{s}_2^2}
\]

Fisher’s information in the input space:

\[
J(w) = \frac{w^T S_B w}{w^T S_w w}
\]

where \( S_B = (m_1 - m_2)(m_1 - m_2)^T \)

\[
S_w = S_1 + S_2 \quad S_i = \sum_{x \in \Omega_i} (x - m_i)(x - m_i)^T, \quad (i = 1,2)
\]

---

W. Kasprzak: EIASR 2. Pattern transformation 57
Two-classes LDA (2)

General solution to $w_o$ is:

$$S_W^{-1}S_B w_o = \lambda w_o$$

Here the expression $S_B w$ is a vector along the direction of $(m_1 - m_2)$ and the scaling of $w$ is not important. Thus:

$$w_o = S_W^{-1}(m_1 - m_2)$$

General LDA (more than two classes, $k > 2$)

The within-class variability matrix:

$$S_W = \sum_{i=1}^{k} S_i = \sum_{i=1}^{k} \sum_{j=1}^{N_i} (x_j - m_i)(x_j - m_i)^T$$

where $N_i$ - number of samples from class $i = 1, ..., k$.

The between-class variability matrix:

$$S_B = \sum_{i=1}^{k} N_i (m_i - \bar{m})(m_i - \bar{m})^T$$

where $\bar{m}$ - the mass centre of all the samples.

General LDA

Definition

The linear Fisher's discriminate function is: $y = W_o x$, where

$$W_o = \arg \max_{W} J(W) \quad \text{and} \quad J(W) = \frac{|W^T S_B W|}{|W^T S_W W|}.$$ 

In LDA we apply the $W_o$ that induces the maximum ratio between the determinants of the between-class variability matrix and the within-class variability matrix of samples.

Solution

The rows of matrix $W_o$ are eigenvectors of the matrix $S_W^{-1} S_B$:

$$S_W^{-1} S_B W_o = \Lambda W_o$$
4. The inverse problem

**Instantaneous-time inverse problem:**
- \( m \) unknown sources (e.g. signals in time) or a set of \( m \)-dimensional data samples:
- \( n \) observed, possibly noisy, different, linear mixtures of the sources \((n \geq m)\):
- the mixing coefficients are some unknown constants.

\[
\begin{bmatrix}
    s_1^T[t] \\
    \vdots \\
    s_m^T[t]
\end{bmatrix}
\begin{bmatrix}
    x_1^T[t] \\
    \vdots \\
    x_n^T[t]
\end{bmatrix}
\]

\[
x[t] = A_s[t] + n[t] = \left( \sum_{i=1}^{m} s_i[t] a_i \right) + n[t]
\]

**Possible solutions**

**Goal:** find the unknown matrix \( A \) and reconstruct the sources.

**Possible solutions:**
1. Independent component analysis
2. Projection pursuit
3. Factor analysis
4. Principal component analysis
Factor analysis, Whitening

Factor analysis (FA) assumes that the sources (called factors) are uncorrelated, of Gaussian distribution and of unit variance: \[ E\{ss^T\} = I \]

The noise components are assumed to be uncorrelated with each other and with the factors: \[ Q = E\{nn^T\} \]

With above assumptions the covariance matrix of the observation is: \[ E\{xx^T\} = R_{xx} = AA^T + Q \]

Assuming \( Q \) is known or can be estimated, FA attempts to solve \( A \) from: \[ AA^T = R_{xx} - Q \]

Without noise the problem simplifies to Whitening: \[ AA^T = R_{xx} \]

It is a specific case of PCA, where every „direction” in space is of unit variance.

ICA for feature extraction

Application of ICA (independent component analysis): a coding scheme - a signal (image block) is represented by a feature vector that consists of mixing coefficients – the block is a mixture of some fixed set of „standard” image blocks (independent components):

\[ x_i = a_{i1} \mathbf{s}_1 + a_{i2} \mathbf{s}_2 + \ldots + a_{im} \mathbf{s}_m \]

Assumptions in ICA:

1. The sources are stochastically independent w.r.t. functions \( f, g \) that capture higher order signal statistics, i.e. for \( y_1, y_2 \) it holds \[ E\{f(y_1)g(y_2)\} = E\{f(y_1)\} \cdot E\{g(y_2)\} \]

2. At most one of the sources is of Gaussian distribution.
Example

Example [Hyvarinen(2000)]

Two independent sources:

\[ p(s_i) = \begin{cases} 
\frac{1}{2\sqrt{3}}, & \text{if } |s_i| \leq \sqrt{3} \\
0, & \text{otherwise} 
\end{cases} \]

Joined distribution \((s_1, s_2)\):

Mixing matrix

\[ A_0 = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \]

Mixture distribution

Example (2)

**Whitening**: a linear transform so that the new mixtures are uncorrelated and have a variance of one.

**PCA**: find orthogonal main directions \(\rightarrow\) no separation.

**ICA**: source separation by generalized independency.
FastICA (1)

FastICA (Hyvarinen, Karhunen, Oja) is a batch method for ICA. It requires that the mixtures are centered and whitened first.

Centering: \( \mathbf{x}_{\text{centered}} = \mathbf{x} - E\{\mathbf{x}\} = \mathbf{x} - \mathbf{m}_x \)

Whitening: \( E\{\mathbf{xx}^T\} = \mathbf{R}_{xx} = \mathbf{U}\Lambda\mathbf{U}^T \)
\( \mathbf{x}_{\text{whitened}} = \Lambda^{-1/2}\mathbf{U}^T\mathbf{x} \)

Then the method iteratively updates the weight matrix \( \mathbf{W} \) – vector-by-vector, while maximizing the non-Gaussianity of the projection \( \mathbf{w}^T \mathbf{x} \) (see next page).

FastICA (2)

A. Initialize a nonzero matrix: \( \mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n] \)

B. Iterate

1) FOR outputs \( p = 1, 2, \ldots, n \) DO

2) Weight update: \( \mathbf{w}_p^+ = E\{\mathbf{x} \ g(\mathbf{w}_p^T \mathbf{x})\} - E\{(g'(\mathbf{w}_p^T \mathbf{x}))\mathbf{w}_p \} \)

where \( g \) is a nonlinear function, \( g' \) – its first derivation.

3) Normalize to unit length: \( \mathbf{w}_p = \frac{\mathbf{w}_p^+}{\|\mathbf{w}_p^+\|} \)

4) A Gram-Schmidt orthogonal and normalization:
\( \mathbf{w}_p' = \mathbf{w}_p - \sum_{j=1}^{p-1} \mathbf{w}_p^T \mathbf{w}_j \mathbf{w}_j^T \quad \mathbf{w}_p = \frac{\mathbf{w}_p'}{\|\mathbf{w}_p'\|} \)

5) IF \( \mathbf{W} \) has not yet converged THEN next iteration of 1)-5).
5. Blind source separation

**BSS** methods solve the inverse problem in a way similar to ICA. But in BSS there is a focus on “on-line” reconstruction of sources.

**Application example**: solving the „cocktail party” problem:

![Diagram of blind source separation](image)

W. Kasprzak: EIASR 2. Pattern transformation

---

**Demixing in BSS**

Demixing in **BSS**: an $m \times n$ separating matrix $W(t)$ is updated so that the $m$-vector, $y(t) = W(t) x(t)$, becomes an estimate, $y(t) \approx s(t)$, of the original independent sources up to scaling and permutation indeterminacy.

If source scaling ($S$) and permutation ($P$) are resolved then:

$$W A = S P I$$
BSS (1)

The Kullback–Leibler divergence measures the dependency among output signals (mutual information):

\[ D(y \| \{y_k\}; W) = \int p(y) \log \left( \frac{p(y)}{\prod_{k=1}^{K} p(y_k)} \right) dy \]

\[ D(y \| \{y_k\}, W) = -H(y; W) + \sum_{k=1}^{K} H(y_k) ; \quad D(y \| \{y_k\}; W) \geq 0 \]

\( H(y, W) \) - the average information of the joint output \( y \). The \( H(y_k) \)-s are entropies of marginal distributions – their sum is constant – only the information related to joint output distribution, \( H(y, W) \), can be optimized here.

The BSS optimization rule:

\[ \text{arg min}_{W} D(y \| y_k; W) \]

BSS (2)

The adaptive separation rule, generated according to the natural gradient:

\[ W(t+1) = W(t) - \eta(t) \frac{\partial D}{\partial W} W^T W \]

is:

\[ W(t+1) = W(t) + \eta(t) \{ I - f[y(t)] \cdot g[y^T(t)] \} \cdot W(t) \]

\( f(y) = [f(y_1), \ldots, f(y_n)]^T \) and \( g(y^T) = [g(y_1), \ldots, g(y_n)] \) are vectors of non-linear activation functions, which approximate higher–order moments of the signals.

If the source has a negative normalized kurtosis value (i.e. it is a sub–Gaussian signal), we choose: \( f(y) = y^3, \ g(y_j) = y_j \).

For a super–Gaussian source with positive kurtosis, we choose: \( f(y) = \tanh(\alpha y), \ g(y_j) = y_j \).
Example

Sound separation: one unknown source, one mixture and one separated signal are shown:

Four unknown sources:

Four mixtures with added noise:

Separated signals:
Measuring the separation quality

(A) If the source signals are known

For every pair (output $Y_i$; source $S_j$) with amplitudes scaled to the interval $<-1, 1>$ compute the $MSE[i, j]$ coefficient - the error of approximating $S_j$ by $Y_i$, $i=1,...,n$; $j=1,...,m$:

$$MSE[i, j] = \frac{1}{N} \sum_{k=0}^{N-1} [S_j(k) - Y_i(k)]^2 = E\{(S_j - Y_i)^2\}$$

where $N$ is number of samples.

A matrix $P = [p_{i,j}]$ is created, with $p_{i,j} = 1 / SNR[i, j]$

The error index:

$$EI(P) = \frac{1}{m} \left( \sum_i \sum_j |\tilde{p}_{i,j} - n| \right) + \frac{1}{n} \left( \sum_j \sum_i |\tilde{p}_{i,j} - m| \right)$$

Every row $i$ of $P$ is scaled: $\tilde{P} = Norm(P)$, such that $\forall i (\max_j (\tilde{a}_{i,j}) = 1)$

Every column $j$ is scaled: $\tilde{P} = NormCol(P)$, such that $\forall j (\max_i (\tilde{a}_{i,j}) = 1)$

Ideal case of perfect separation:

- $P$ becomes a permutation matrix.
- Only one element in each row and column equals to unity, and all the other elements are zero.
- Then the minimum $EI = 0$.

Measuring the quality (2)

(B) If the mixing matrix $A$ is known

The error index for the set of separated sources is given as:

$$EI(P) = \frac{1}{m} \left( \sum_i \sum_j |\tilde{p}_{i,j} - n| \right) + \frac{1}{n} \left( \sum_j \sum_i |\tilde{p}_{i,j} - m| \right)$$

The entries $p_{i,j}$-s of matrix $P$, $P = W A$, are normalized along rows ($i=1, ..., n$) (for the first expression) or columns ($j=1,...,m$) in $P$, such that: $\forall i (\max_j (\tilde{a}_{i,j}) = 1)$ or $\forall j (\max_i (\tilde{a}_{i,j}) = 1)$
Measuring the quality (3)

(C ) If both the sources and mixing matrix are unknown

The normalised mutual correlation coefficients are computed

\[ r_{i,j} = \frac{f(y_i) \cdot g(y_j)}{|f(y_i)||g(y_j)|} \]

for every pair of output signals \( y_i \) and \( y_j \), giving the square matrix: \( P=[r_{i,j}] \).

The error index for the set of separated sources is computed as:

\[
EI(\tilde{P}) = \frac{1}{n} \left( \sum_i \sum_j |r_{i,j}| - n \right)
\]
1. Numeric classifiers

Optimization criteria for classifiers
1. Minimum computational complexity $\Rightarrow$ linear discriminate classifiers
2. Minimum of misclassification probability $\Rightarrow$ Bayes classifier
3. Ability automatically to learn a decision rule $\Rightarrow$ multi-layer perceptron
4. Ability to generalise knowledge for not available samples $\Rightarrow$ “Support Vector Machine”
5. Minimization of general risk $\Rightarrow$ ???

Decision theory approach

A. Make assumptions about the analysed domain – for example learn the prior $pdf$-s.
B. Define the costs and risk of misclassification (decision).
C. Define a decision rule, which minimises this risk.

Stochastic classifier - the analysed domain and decision rule in terms of stochastic variables.

Let us observe:
• Each decision induces individual costs.
• After many decisions the average (expected) costs can be estimated – the risk of misclassification.
• The calculation of average values requires the availability of full statistic information ($pdf$-s).
Risk minimization

(A) Assumptions about the analysed domain:

• Pdf of observation $c$ given class $k$, $p(c \mid k)$.
• Prior probabilities $\{ p_k \mid k=1,\ldots,K \}$ of patterns from classes.

(B) Calculation of risk $V(kc)$ - a pattern $kc$ from class $k$.

1. **Costs** of misclassification: $\{ r_{ik} = r(i \mid k) \mid i,k=1,2,\ldots,K \}$;
2. Obtain the probability of pattern from class $k$: $p(kc \mid k) p_k$
3. The risk of pattern $kc$: $V(kc) = \sum_k \sum_i (p_k p(kc \mid k) r_{ik})$.

(C) Decision rule

1. **Minimise** the risk $V$. If a binary cost function is used:
   
   $$r_{kk} = 0, \quad r_{ik} = 1, \quad \text{for} \quad i \neq k, \quad \text{and} \quad i, k = 1, \ldots, K.$$  
   
   then we need to maximize $p(k \mid kc)$.

2. **Select** class $k$ with highest $p(k \mid c)$:
   
   $$i = \arg_k \max p(k \mid c).$$

2. Potential function-based classifier

- **Potential functions** as class distributions in feature space;
- All functions from the same **parametric family** of functions.

Linear potential function family:

$$d_i(c,a) = \{a_i^T \varphi(c) \mid a_i \in \mathbb{R}^{(n+1)} \}, (i=1,\ldots,m)$$

$$\varphi(c) = (1,c_1,c_2,\ldots,c_n)^T$$

**Linear potential function-based classifier, two class-case:**

Get potentials:

$$d_1(c,a^{(1)}) = a_0^{(1)} + \sum_{i=1}^n a_i^{(1)} \cdot c_i \quad d_2(c,a^{(2)}) = a_0^{(2)} + \sum_{i=1}^n a_i^{(2)} \cdot c_i$$

Decision rule: $d(c) = d_1(c,a^{(1)}) - d_2(c,a^{(2)})$,

If $d(c) \geq 0$ then select the class $\Omega_1$ else class $\Omega_2$.
Example

Binary classifier for a 2-D feature vector – the **linear decision rule** corresponds to two half-planes separated by the line: \( d_1(c) = d_2(c) \).

Learning the potential function-based classifier

Linear classifier for \( K \) classes:

\[
\zeta(c) = \arg \max_k d_k(c, a^{(k)}) = \arg \max_k (a_0^{(k)} + \sum_{i=1}^n a_i^{(k)} \cdot c_i)
\]

**Learning:**

1. Let \( \mathbf{C} \) be a matrix of size \( N \times 3 \), built from \( N \) samples, where \( N_k \) samples are from class \( k \).

2. For every class \( \Omega_k \) define the equation system:

\[
\mathbf{e} + \mathbf{D}^{(k)} = \mathbf{C} \mathbf{a}^{(k)},
\]

where \( \mathbf{D}^{(k)} = [d^{(k)}_1, d^{(k)}_2, \ldots, d^{(k)}_N]^T \) and \( d_k(c, a^{(k)}) = 1 \), if \( c \in \Omega_k \), or \( d_k(c, a^{(k)}) = -1 \), if \( c \notin \Omega_k \); and \( \mathbf{e} = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N]^T \) is an error vector.

\( \rightarrow \) to be continued on next page.
Learning the classifier (2)

3. The „least square error” optimization problem is:

\[ a_o = \arg \min_a \sum_{i=1}^{N} \varepsilon_i^2(a) \]

The solution leads to a system:

\[ X^{(k)} = B^{(k)} a_o^{(k)}, \]

where

\[ X^{(k)} = C^{(k)T} D^{(k)}, \quad B^{(k)} = C^{(k)T} C^{(k)}. \]

This can be solved directly as:

\[ a_o^{(k)} = (C^{(k)T} C^{(k)})^{-1} C^{(k)T} X^{(k)}. \]

3. Bayes classifier

Stochastic distributions are needed:
- prior class pdf-s \( p(\Omega_k) \), \( k = 1, \ldots, K \)
- conditional pdf-s \( p(c \mid \Omega_k) \).

Decision rule:

\[ \zeta(c) = \arg \max_{\lambda} p(\Omega_{\lambda} \mid c) = \arg \max_{\lambda} p(\Omega_{\lambda}) p(c \mid \Omega_{\lambda}). \]

Learning the probability distributions:

1. \( p(\Omega_k) \) are estimated as relative frequencies of classes in the learning sample set;
2. \( p(c \mid \Omega_k) \): non-parametric, e.g. histograms; or parametric - assume a density function family and estimate the parameters from the learning samples.
Learning a Bayes classifier

Non-parametric pdf

Parametric density

ML („maximum likelihood”) estimator: for \( \Omega_k \) specify parameter set \( \theta^{(k)} \) which maximizes the probability of given observations

\[
\max_{\theta^{(k)} \in \Theta} \sum_{c_j \in C^{(k)}} P(c_j | \theta^{(k)})
\]

Solve:

\[
\frac{\partial}{\partial \theta_i} \log \sum_{c_j \in C^{(k)}} P(c_j | \theta^{(k)}) = 0 \quad , \quad \theta_i \in \theta^{(k)}
\]

Minimum-distance classifier

This is a specific form of the Bayes classifier for Gaussian pdf-s if it uses the Euclidean metrics.

Select the class of minimum distance: \( d(\Omega_k , c) = \ln p(\Omega_k | c) \).

Then in special case \( (\Sigma_1 = \Sigma_2 = I) \) the Bayes condition:

\[
p(\Omega_1) p( c | \Omega_1 ) > p(\Omega_2) p( c | \Omega_2 )
\]

is equivalent to:

\[
\ln p(\Omega_1) - \frac{1}{2} (c - \mu_1)^T \Sigma_1^{-1} (c - \mu_1) > \ln p(\Omega_2) - \frac{1}{2} (c - \mu_2)^T \Sigma_2^{-1} (c - \mu_2)
\]

With, \( \Sigma_1 = \Sigma_2 = I \) , and the maximum likelihood rule,

\[
p(\Omega_1) = p(\Omega_2), \text{ it simplifies to: } |c - \mu_1|^2 < |c - \mu_2|^2
\]

This is the rule of a geometric minimum-distance classifier that uses the Euclidean metrics.
4. The \textbf{k-NN classifier}

Learning.
A set of feature vectors: \( c^i \in C = \{ c^1, c^2, ..., c^n \} \), where each sample belongs to class \( \zeta(c^i) \).

Learning a nearest neighbour-classifier means simply to \textbf{store} all samples. The set of points in the feature space for which the nearest neighbour among the learning set is equal to \( c^i \) is called a \textbf{Voronoi cell} of \( c^i \).

Decision rule \( \zeta(\cdot) \):
select the most frequent class of the \( k \) nearest neighbours of \( c \).

5. \textbf{Support Vector Machine}

\textbf{SVM} is a discriminate-like binary classifier, considered when proper class distributions over the feature space are difficult to be established (\textbf{limited} set of data samples).

SVM learning (or design): to find an optimal hyper-plane (a linear SVM) or hyper-surface (for non-linear SVM) that separates the two areas in feature space representing two classes.

The classification with SVM consists of many binary decisions between two classes (or class groups) (denoted as „+1”, „-1”).

The \textit{Vapnik–Chervonenkis dimension} (VC) \( h \) is a measure for the set of separating functions. In case of 2 classes \( h \) determines the maximum amount of patterns that can be separated on all possible ways, i.e. this number of separations is \( 2^h \).
**VC dimension**

**Thesis.** The Vapnik-Chervonenkis dimension of the set of oriented hyper-planes in the $\mathbb{R}^n$ space is $h = n + 1$.

**Example.** Three points on a plane (2-D points) can be separated by an „oriented” line into all possible two sub-sets, the number of such partitions is $2^3 = 8$. This is no longer true for 4 points; hence for 2-D points ($n=2$) the VC dimension is $h=3$.

VC dimension gives the number of „supporting vectors”, the number of samples needed to define the separating hyperplane.

---

**SVM for a linearly separable set**

**Example.** *The separating hyper-plane $H^*$. There could be other separating planes (e.g. $H'$) which are not optimal.*

The optimization criterion: to maximize the distance of the nearest „support vectors” (or to minimize $|a|^2$).
**SVM optimization problem**

**Definition.** The SVM for a linearly separable set of samples from 2 classes is determined by a hyper-plane

\[ H^* : d_{\tilde{a}}(c) = c^T a + a_0 = 0 \]

that satisfies the goal:

\[ \min_{\tilde{a} \in \mathbb{R}^{n+1}} f(a) = \min_{\tilde{a} \in \mathbb{R}^{n+1}} \left( \frac{1}{2} |a|^2 \right) \]

and \( N \) additional constraints \((j = 1, \ldots, N)\):

\[ y_j (c^T a + a_0) - 1 \geq 0, \quad \forall c \in \omega \]

This optimization problem is called **convex quadratic programming** with inequality-based linear constraints.

The decision rule is:

\[ \zeta(c) = \begin{cases} \Omega_1, & \text{if } d_{\tilde{a}}(c) \geq 0 \\ \Omega_2, & \text{if } d_{\tilde{a}}(c) < 0 \end{cases} \]

**Support vectors**

The constraints are included into the goal function using positive-valued Lagrange multipliers: \( \mathcal{g} = (g_1, \ldots, g_N)^T \).

A modified **goal** – the **Lagrange** function:

\[ L(\tilde{a}, \mathcal{g}) = \frac{1}{2} |a|^2 - \sum_{j=1}^{N} g_j (y_j (c^T a + a_0) - 1) \]

**Support vectors** are the only samples needed for the hyper-plane; their additional constraints are active and they have non-zero Lagrange multipliers, \( g_j > 0 \).

\[ H^* : d_{\tilde{a}}(c) = \sum_{j=1}^{N} g_j y_j (c^T c) + a_0 = 0 \]

The hyperplane is a **weighted scalar product** of the support vectors.
Linear SVM under noise

Linear separation under noisy condition: adding a “penalty” component in the SVM’s goal function - covering wrongly separated samples.

Example. Error distances $\varepsilon_5$ and $\varepsilon_6$ (for samples of class +1) and $\varepsilon_7$ and $\varepsilon_8$ (samples of class -1).

The dual problem

The modified goal function:

$$
\min_{\tilde{a} \in R^n} f(a) = \min_{\tilde{a} \in R^n} \left( \frac{1}{2} |a|^2 + C \sum_{j=1}^{N} \varepsilon_j \right)
$$
subject to

$$
y_j (\mathbf{c}^T \mathbf{a} + a_0) \geq 1 - \varepsilon_j, \quad \forall j \in \omega, \varepsilon_j \geq 0
$$

$C$ is a constant, controlling the influence of the noisy component onto the goal function.

The dual problem: $\max_{\mathcal{G}} L_D(\mathcal{G}) = \max_{\mathcal{G}} \left[ \inf_{\tilde{a}} L(\tilde{a}, \mathcal{G}) \right]$

with

$$
L_D(\mathcal{G}) = \sum_{j=1}^{N} \mathcal{G}_j - \frac{1}{2} \sum_{j=1}^{N} \sum_{k=1}^{N} \mathcal{G}_j \mathcal{G}_k y_j y_k (\mathbf{c}^T \mathbf{c}) = -\frac{1}{2} \mathcal{G}^T \mathbf{Q} \mathcal{G} + \mathbf{i}^T \mathcal{G}
$$

where

$$
\mathbf{Q}_{jk} = y_j y_k \mathbf{c}^T \mathbf{c} \quad ; \quad \mathbf{i} = [1,1,...,1]^T
$$

subject to

$$
0 \leq \mathcal{G}_j \leq C \quad ; \quad 0 = \sum_{j=1}^{N} \mathcal{G}_j y_j = \mathbf{y}^T \mathcal{G}
$$
Solving the SVM problem

Both the primary and the dual form of the SVM optimization problem are instances of the general quadratic programming problem (for example see the quadprog function in MATLAB).

There exists traditional methods, like Newton method or active set method, for solving the quadratic programming problem.

But with many hundreds or thousands training samples traditional methods cannot be directly applied.

An efficient learning algorithm that solves the dual problem for SVM is “Sequential minimal optimization” (SMO) (J.C.Platt).

Kernel SVM

Linear separators in a high-dimensional feature space $F(c)$, by replacing $c$ and $j c$ in $H^*$ equation with $F(c)$ and $F(j c)$,

$$H^* : d_a(c) = \sum_{j=1}^{N} \theta_j y_j (F(c)^T F(j c)) + a_0 = 0$$

Example:

(a) The true decision boundary is: $x_1^2 + x_2^2 \leq 1$. (b) After mapping into a three-dimensional feature space: $(x_1^2, x_2^2, \sqrt{2} x_1 x_2)$. 
6. MLP

McCulloch-Pitt’s model of a single neuron:

\[
\text{Inputs: } (x_1, x_2, \ldots, x_n)
\]

Input function: \( y = \sum_{i=1}^{n} (w_i \cdot x_i) - w_0 \cdot 1 \), where \( w_0 \) - bias weight

Activation function: \( z = \theta(y) \)

Output: \( z \)

Single feed-forward layer

In ANN (artificial neural networks) the neurons are organised into layers.

In a feed-forward (perceptron) network the input signals are sent from the input to the output via excitatory links.

\[
y = W \cdot x
\]

\[
z = \theta(y)
\]
Activation functions

Single step function: \( z = 1 \), if \( y_i > y_k \), where \( y_k \) - a fixed threshold.

Sigmoid function: 
\[
\theta(y) = \frac{1}{1 + \exp(-\beta y)}
\]

Example. Activation functions: (left) step function, (right) sigmoid function.

Sigmoid function

The derivative of a sigmoid function is:
\[
\frac{dz}{dy} = z(1 - z)
\]

Example. Sigmoid function with different parameter \( \beta \).
Multi-layer perceptron

**MLP:** some number (at least two) of feed-forward layers arranged in a cascade. The transformation function of each layer $l (=1,2,...)$: $z^{(l)} = \theta(W^{(l)}z^{(l-1)} - w_0^{(l)})$, where $z^{(0)} = x$.

Example. A **MLP with 3 layers (2 hidden layers).**

---

**Supervised learning of MLP**

*The Widrof-Hoff rule (delta rule)* – the weight of incoming link between $i$-th neuron and $j$-th input is strengthened in proportion to the difference between required and real activation:

$$\Delta w_{ij} = \mu(s_i - z_i)x_j$$

where $s_i$ is the required activation of the $i$-th neuron, and $\mu$ - a learning coefficient.

The **error back-propagation rule** in MLP is an extension of the delta rule. An update of weights in layer $l$ ($l = L, L-1, ..., 1$):

$$\Delta W^{(l)} = \mu \cdot d^{(l)} \cdot z^{(l-1)T}, \quad z^{(0)} = x$$

where $d^{(l)}$ is the correction vector for the $l$-th layer.

A single learning iteration starts form the last layer ($l = L$) and it proceeds back-to-back, layer-by-layer, until $l=1$. 

---

W. Kasprzak: EIASR 3. Pattern classification
Backpropagation learning

The correction values are set as follows:
For last layer the required output vector \( s \) is available. Hence
\[
d^{(L)} = (s - z^{(L)}) \ast \begin{bmatrix} \frac{\partial z_i}{\partial y_i} \end{bmatrix}
\]
For hidden layers the correction vector is projected back. Hence, for \( l = L-1, \ldots, 1 \):
\[
d^{(l)} = (W^{(l+1)})^T d^{(l+1)} \ast \begin{bmatrix} \frac{\partial z_i}{\partial y_i} \end{bmatrix}
\]
"\( \ast \)" - an element-by-element multiplication of two vectors.

For a \textbf{sigmoid} activation function the corrections are:
\[
d^{(L)} = (s - z^{(L)}) \ast \begin{bmatrix} 1 - z^{(L)} \end{bmatrix} * z^{(L)}
\]
\[
d^{(l)} = (W^{(l+1)})^T d^{(l+1)} \ast \begin{bmatrix} 1 - z^{(l)} \end{bmatrix} * z^{(l)}
\]

7. Pattern clustering

\textbf{Unsupervised learning} procedures use unlabeled samples.

\textbf{„k-means” clustering}
1. Init \( k \) cluster centers: \( \mu^{(i)} \).
2. FOR each observation \( c_j \) DO
associate it with the closest cluster center, that is, assign \( \xi(c_j) \leftarrow \mu^{(i)} \), where \( d(\mu^{(i)}, c_j) = \min_k d(\mu^{(k)}, c_j) \), for some distance function \( d \) (e.g. Euclidean distance).
3. FOR each cluster \( i \) (with \( n^{(i)} \) assigned observations) DO
estimate the mean of the observations assigned with cluster as:
\[
\mu^{(i)} = \frac{1}{n^{(i)}} \sum_{j, \xi(c_j) = i} c_j
\]
4. REPEAT steps 2 and 3 given number of times.
EM clustering

The expectation-maximization (EM) algorithm is a general estimation technique, dealing with missing data. If applied for clustering the EM algorithm gives an maximum-likelihood estimate of a Gaussian mixture:

\[ p(c \mid \theta) = \sum_{i=1}^{K} \alpha^i N(c \mid \mu^i, \Lambda^i) \]

where \( N(.) \) denotes a Gaussian pdf, and \( \theta \) - the parameter set that needs to be estimated: \( \theta = \{ \alpha^i, \mu^i, \Lambda^i \mid i=1,2,\ldots,K \} \).

**E-step:** FOR each sample \( c_j \) and

FOR each class \( i \) DO:

**M-step:** FOR each class \( i \) DO:

\[
\alpha^i = \frac{\sum_j P^i_j}{\sum_i \sum_j P^i_j}, \quad \Lambda^i = \frac{\sum_j P^i_j \mu^i}{\sum_i \sum_j P^i_j}, \quad \mu^i = \frac{\sum_j c_j P^i_j}{\sum_j P^i_j}
\]

8. Ensemble learning

Many classifiers or experts are generated and combined to solve a particular classification or decision problem:

1. **Boosting** – combining weak binary classifiers into a strong one. Each iteration of boosting creates three weak classifiers: the first classifier C1 is trained with a random subset of the training data. C2 is trained on a set only half of which is correctly classified by C1, and the other half is misclassified. C3 is trained with instances on which C1 and C2 disagree. The 3 classifiers are combined through a three-way majority vote.

→ **AdaBoost.M1 (adaptive Boosting)** – extension for multi-class classification, most popular ensemble classifier (Freund & Shapire, 1995)

2. **Mixture of experts** (Jacobs & Jordan, 1991-94)
AdaBoost

**AdaBoost** is an algorithm for constructing a "strong" classifier as a linear combination of \( T \) "weak" classifiers:

\[
F(x) = C[f(x)] = C \left[ \sum_{t=1}^{T} w_t \cdot h_t(x) \right]
\]

where \( h_t(x) \) is the classification ("hypothesis") of sample \( x \) given by the \( t \)-th weak classifier, \( w_t \) are weights and \( C \) is the rule of the "strong" classifier.

The set of "weak" classifiers, \( H = \{h_i(x)\} \), is potentially infinite.

Training data samples are drawn from a distribution that is iteratively updated, such that subsequent classifiers focus on increasingly difficult instances. Previously misclassified instances are more likely to appear in the next bootstrap sample. The classifiers are combined through weighted majority voting.

---

**AdaBoost.M1 (1)**

**INPUT**: classes: \( \Omega=\{\Omega_1, \Omega_2, \ldots, \Omega_L\} \); training samples: \( S = \{x_1, x_2, \ldots, x_n\} \), with labels \( y_i \in \Omega, i=1,2,\ldots,n \); number of weak classifiers: \( T \).

**TRAINING**

INIT distribution: \( D_1(i) = \frac{1}{n}, i=1,2,\ldots,n \)

FOR \( t=1,2,\ldots,T \) DO

1. Select current training subset \( S_t \) according to \( D_t \).
2. **WeakLearn**: train on \( S_t \) and return the "weak" classifier, \( h_t: X \rightarrow \Omega \), with smallest error: \( \varepsilon_t = \sum_{i=1}^{n} I(h_t(x_i) \neq y_i) \cdot D_t(i) \)
   
   IF \( \varepsilon_t > 1/2 \) THEN STOP
3. Calculate normalized error: \( \beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t} \)
4. Update distribution \( D_t \):
   
   \[
   D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \beta_t, \text{ if } h_t(x_i) = y_i \quad \text{or} \quad D_{t+1}(i) = \frac{D_t(i)}{Z_t}, \text{ if } h_t(x_i) \neq y_i
   \]

---

W. Kasprzak: EIASR 3. Pattern classification

109
AdaBoost.M1 (2)

Remark: $Z_t$ is a normalization coefficient that makes $D_{t+1}$ a proper distribution.

CLASSIFICATION by a weighted majority voting:
Given a new unlabeled sample $s$:
1. Obtain total vote received by each class,
   \[ v_j = \sum_{t=1}^{T} I(h_t(s) \neq \Omega_j) \cdot \log(1/ \beta_t), \quad j = 1, \ldots, L \]
   i.e. the „weak” weights are: \[ \log(1/ \beta_t) \]
2. Rule $C$: select the class $\Omega_j$ which receives the highest total vote $v_j$

Mixture of Experts (1)

Several experts (classifiers) are learned. The $i$-th expert produces its output:
\[ o_i(x) = f(W_i x) \]
where $W_i$ is a weight matrix and $f(.)$ - a fixed nonlinearity.

The outputs of experts are combined through a (generalized) linear rule:
\[ y = o(x) = \sum_{k=1}^{N} g(x, v_k) o_k(x) \]

The weights of this combination are determined by a gating network:
\[ g(x, v_i) = \frac{e^{\xi_i}}{\sum_{k=1}^{N} e^{\xi_k}}; \quad \xi_i = v_i^T x \]
Mixture of Experts (2)

The classification step in the „mixture of experts” approach can be expressed in stochastic terms as the maximization of posterior probability:

\[ P(y \mid x, \Psi) = \sum_{k=1}^{N} g(x, v_k) P(y \mid x, W_k) \]

where \( \Psi \) is the set of all parameters (all expert- and gating weights).

The parameters are typically trained using the expectation maximization (EM) algorithm.

Let the training set is given as: \( \{(x_t, y_t) \mid t=1,2,\ldots,T\} \). In the E-step, in the \( s \)-th epoch, for all the training data following posterior probabilities are computed:

\[
p_i(t) = \frac{g(x_t, v_i^{(s)}) P(y_t \mid x_t, W_i^{(s)})}{\sum_{k=1}^{N} g(x_t, v_k^{(s)}) P(y_t \mid x_t, W_k^{(s)})}, \quad i=1,2,\ldots,N
\]

Mixture of Experts (3)

In the M-step following maximization problems are solved:

\[
W_i^{(s+1)} = \arg \max_{W_i} \sum_{t=1}^{T} p_i(t) \log P(y_t \mid x_t, W_i)
\]

\[
V^{(s+1)} = \arg \max_{V} \sum_{t=1}^{T} \sum_{k=1}^{N} p_i(t) \log g(x_t, v_k)
\]

where \( V \) is the set of all the parameters in the gating network.
1. Scene acquisition

Perspective projection
A simple „pinhole” model of the camera:

\[ p' = (-x_p, -y_p) \] – reversed image point, \( P = (x_c, y_c, x_c) \) – scene point (in camera coordinates), \( f \) – focal length.

\[ x_p = s_x \frac{f \cdot x_c}{z_c} \]

\[ y_p = s_y \frac{f \cdot y_c}{z_c} \]

\( s_x, s_y \): scaling of scene-to-image units
Perspective vs parallel projection

Parallel projection – if $f$ tends to infinity:

$$x_p = \lim_{f \to \infty} s_x \frac{f \cdot x_c}{f + z_c} = s_x x_c$$

$$y_p = \lim_{f \to \infty} s_y \frac{f \cdot y_c}{f + z_c} = s_y y_c$$

Perspective projection vs. parallel projection

Vector operations

3-D vectors:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Inner product of two vectors:

$$\| \mathbf{u} \| = \sqrt{u_1^2 + u_2^2 + u_3^2}$$

$$\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\| \mathbf{u} \| \| \mathbf{v} \|}$$

Cross product of two vectors

$$\hat{\mathbf{u}} = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}$$

$$\mathbf{u} \times \mathbf{v} = \hat{\mathbf{u}} \mathbf{v}$$
Camera parameters

A transformation of scene point \( P \) onto pixel \( p \) depends on \textbf{camera parameters} of two types:

1. **Extrinsic** parameters \( \rightarrow \) rotation \( R \), translation \( T \)
2. **Intrinsic** parameters \( \rightarrow \) projection \( K_F \), camera-to-image \( K_s \)

**Extrinsic camera transformation**

Let \( P_w = [x_w, y_w, z_w] \) be the world coordinates and \( P_c = [x_c, y_c, z_c] \) the camera coordinates of some scene point \( P \). Then:

\[
P_c = RP_w + T
\]

**Intrinsic camera transformation**

\[
\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K_f P_c = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}
\]

\[
P = \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = K_s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

Homogeneous coordinates

A 3-D point \( P \) \( \leftrightarrow \) 4-D \textbf{homogeneous} coordinates \( P_h \):

\[
P = [X, Y, Z]^T \leftrightarrow P_h = [kX, kY, kZ, k]^T, \quad k \neq 0.
\]

Translate a point by vector \( [t_x, t_y, t_z]^T \):

\[
T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]

Scaling of coordinates by \( [s_{x'}, s_{y'}, s_{z'}]^T \):

\[
S = \begin{bmatrix} s_{x'} & 0 & 0 \\ 0 & s_{y'} & 0 \\ 0 & 0 & s_{z'} \\ 0 & 0 & 0 \end{bmatrix}
\]

Rotate the coordinate system:
- around axis \( Z \) by angle \( \theta \),
- around axis \( X \) by angle \( \alpha \),
- around axis \( Y \) by angle \( \beta \)

\[
R_\theta = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_\alpha = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_\beta = \begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]
Homogeneous coordinates (2)

**Perspective** transformation of homogeneous coordinates: Reverse transformation is not unique!

\[
\Psi_f = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/f & 0 \\
\end{bmatrix}
\]

Let us observe:

\[
p_h = \begin{bmatrix}
x_p \\
y_p \\
f \\
1 \\
\end{bmatrix}
= \text{norm}(\Psi_f \cdot P_h) = \text{norm}
\begin{bmatrix}
x_c \\
y_c \\
z_c \\
z_c/f \\
\end{bmatrix}
= \begin{bmatrix}
f x_c/z_c \\
f y_c/z_c \\
f \\
1 \\
\end{bmatrix}
\]

The transformation from camera to pixel units

The operations of pixel scaling, skewing and shift of optical center can be represented by a single matrix for operation on homogeneous coordinates.

W. Kasprzak: EIASR

2. Camera calibration

**Example.** Extrinsic parameters in \( R, T \):
- \( \alpha \): rotation of axes around global axis \( OZ \);
- \( \beta \): rotation of axes around global axis \( OY \);
- \( \theta \): rotation of axes around global axis \( OX \);
- \( G \): translation of global system to camera fixture system;
- \( C \): translation of camera fixture system to camera system;

**Intrinsic parameters in \( \psi_f, \psi_s \):**
- \( f \): the camera’s focal length,
- \( s_x, s_y \): the image-to-pixel unit scaling,
- \( o_x, o_y \): the camera origin to image origin shift.
Example (2)

The transformation of a point $P_w$, given in world coordinates, into a pixel $p$:

$$ p = \left( \Psi_s \cdot \Psi_f \cdot T_{-C} \cdot R_\theta \cdot R_\beta \cdot R_\alpha \cdot T_{-G} \right) \cdot P_w = A \cdot P_w $$

Remark: in this example we have defined a translation of the coordinate system, while in above equation a point is moved. A transformation of a point is a dual operation to the transformation of a coordinate system.

Auto-calibration of the camera

The goal is to estimate the combined matrix $A$ for the scene-to-image transformation, based on some number of pixel-to-scene point correspondences.

$$ p = \begin{bmatrix} kx \\ ky \\ kz \\ k \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = A \cdot P_w $$

Auto-calibration

Matrix $A$ consists of 12 parameters, but only 8 are independent (the 3-d row linearly depends on the 4-th row (by $f$)).

$$ p = \begin{bmatrix} kx \\ ky \\ kz \\ k \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = A \cdot P_w $$

As $p_1 = xp_4$, and $p_2 = yp_4$, we get a system of 3 linear equations:

$$ xp_4 = a_{11} X + a_{12} Y + a_{13} Z + a_{14} $$
$$ yp_4 = a_{21} X + a_{22} Y + a_{23} Z + a_{24} $$

As $p_4 = a_{41} X + a_{42} Y + a_{43} Z + a_{44}$

$(x,y)$ and $(X,Y,Z)$ represent measurable data while $p_4$ is not known. It needs to be eliminated from the system.
Auto-calibration (2)

After elimination of $p_4$ from the 1st and 2nd row we get 2 equations with 12 unknowns:

\[ a_{11}X + a_{12}Y + a_{13}Z - a_{41}X - a_{42}Y - a_{43}Z - a_{44}x + a_{14} = 0 \]
\[ a_{21}X + a_{22}Y + a_{23}Z - a_{41}yX - a_{42}yY - a_{43}yZ - a_{44}y + a_{24} = 0 \]

**Auto-calibration algorithm**

1. Detect $m \geq 6$ image points \{ $p_i = (x_i, y_i)$ $(i=1,2,...,m)$ \}, which are projections of known 3-D points (in world coordinates) - $P_i = (X_i, Y_i, Z_i)$ $(i=1,2,...,m)$.
2. For every pair of points $(p_i, P_i)$ make two equations of above form – in total $M \geq 12$ equations with 12 unknowns $a_{11} - a_{44}$.
3. Solve the equation system $M$ by an LSE approach.

---

3. Color spaces

**Color spaces** that are based on psychological observations on human perception:

- HLS (Hue, Lightness, Saturation),
- HSV (Hue, Value, Saturation).

**Technical systems** represent color as a mixture of three basic vectors in the color space:

- A typical color system for computer monitors - three basic colors: red, green, blue (the RGB image, where R,G,B are coefficients from the interval $[0, 1]$). An additive positive-oriented system - "white" is the sum of its components.
- Typical color printings on printers - the CMY (Cyan, Magenta, Yellow) system. An additive "negative" model - "black" is the sum of components:

\[ [C, M, Y] = [1, 1, 1] - [R, G, B]. \]
Color spaces

- Typical analog television broadcast - YIQ (Y - luminance, I, Q - chromatic components: red and red-blue). A specific sensitivity of the human eye to the green color allows to represent green mostly by Y.

\[
\begin{bmatrix}
Y \\
I \\
Q
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
0.60 & -0.28 & -0.32 \\
0.21 & -0.52 & 0.31
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

- In digital media - YUV (or Y \(C_b\) \(C_r\)), where \(C_b\), \(C_r\) - distances of the color from "gray" color along blue and red axes.

\[
\begin{bmatrix}
Y \\
C_b \\
C_r
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.1687 & -0.3313 & 0.5 \\
0.5 & -0.4187 & -0.0813
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\quad \begin{bmatrix}
Y \\
U \\
V
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.1471 & -0.2889 & 0.436 \\
0.6149 & -0.515 & -0.100
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

Note: R, G, B are coefficients from [0, 1]. Y also takes values from [0, 1], but \(C_b\), \(C_r\) (or U, V) take values from [-0.5, 0.5].

W. Kasprzak: EIASR 4. Image pre-processing

Colour calibration

Ideal colours (blue, green, red, yellow, magenta, white, dark), specific "skin" colours ("dark skin" and "light skin"), etc. are defined in the YUV space. They are applied to make a colour calibration of the image.

```c
const unsigned char kIdealMcBethYUV[] =
{
    {88, 114, 146}, // Dark Skin
    {161, 110, 151}, // Light Skin
    ...,
    {140, 69, 182}, // Orange
    ...
    {66, 176, 113}, // Blue
    {117, 100, 95}, // Green
    {85, 111, 193}, // Red
    {191, 35, 161}, // Yellow
    {120, 143, 175}, // Magenta
    {98, 166, 57}, // Cyan
    ...,
    {241, 128, 128}, // White
    {201, 128, 128}, // Neutral 0.8
    {161, 128, 128}, // Neutral 0.65
    {121, 128, 128}, // Neutral 0.5
    {83, 128, 128}, // Neutral 0.35
    {49, 128, 128}, // Black
    ...,
};
```

Colour plane U-V for mid-value Y=0.5 (of 1) (or 128 of 255)
Y-based color normalization

In theory an Y-based normalization of the color components U, Y is not required. But in practice this can help to limit the variability of “dark” and “light” versions of the same color.

Assume 8 bits per color per pixel representation – i.e. values from [0, 255].

\[
\begin{align*}
Y_p' &= 128; \\
U_p' &= U_p + \kappa_{pU} (Y_p' - Y_p); \\
V_p' &= V_p + \kappa_{pV} (Y_p' - Y_p);
\end{align*}
\]

E.g. for a skin color: \(\kappa_{pU} = 1.135\) and \(\kappa_{pV} = 1/\kappa_{pU}\)

Example

The detection of skin colour in the image:
4. Image enhancement

Low contrast images can be enhanced by "histogram stretching" or "histogram equalization".

"Stretching" of histogram \( H \)

1. Find \( a = \min(H, A) \) and \( b = \max(H, A) \), such that \( A\% \) of pixels have lower value than \( a \) and \( A\% \) of pixels have higher value than \( b \) (e.g. \( A=0.05 \)).
2. Transform the pixel value \( w \) to a new digital value in the interval \([0, 1, ..., D_m]\) as:

\[
f(w) = \begin{cases} 
0 & \text{if } w < a \\
D_m \cdot \frac{w-a}{b-a} & \text{if } a \leq w \leq b \\
D_m & \text{if } w > b 
\end{cases}
\]

Histogram equalization

The purpose of histogram equalization is to make every pixel value (nearly) equally probable:

\[
a = f(w) = \text{round}\left[D_m \cdot \frac{1}{\text{PixNum}} \sum_{i=0}^{w} H(i)\right]
\]

where \( H(i) \) is the histogram value for pixel value \( i \), \( \text{PixNum} \) – total pixel number, \([0, 1, ..., D_m]\) is the interval of digital levels of variable \( w \) in the original image, \text{round}[] – specifies the rounding operation to nearest digital level.

Remark: perfectly flat histograms are seldom obtained when working with digital variables. Equalization usually leads to a reduction of pixel value levels in the transformed image if compared to the original one.
Example

Image and its histogram before and after histogram equalization:

5. Image binarization

Problem: how to set the threshold for image binarization – to separate the foreground object pixel from the background?

Solution idea: the image histogram is approximated by a weighted sum of two normal (Gaussian) distributions. The border between distributions determines the threshold value:

\[ p(l) = \alpha N(l, m_1, \sigma_1) + (1 - \alpha) N(l, m_2, \sigma_2) \]
The Otsu method

1. FOR all possible thresholds $\theta$;
   obtain the between-class variance as:
   \[
   \sigma(\theta)_B^2 = P_1(\theta)(1 - P_1(\theta))(m_1(\theta) - m_2(\theta))^2
   \]
   where $P_1(\theta) = \frac{\sum_{l=1}^{\theta} h(l)}{\sum_{l=1}^{L} h(l)}$.

2. Select $\theta$ such that $\sigma(\theta)_B^2$ is of maximum value.

There exists a finite number of threshold positions only. Hence, the above algorithm always terminates.

6. Pattern normalization

An image object can appear of different size and orientation and nevertheless we need to recognize it.

Example: different letters W, but the same class of letter.

Example: normalization of image size
Pattern normalization (1)

Pattern transformation in image space (translation, scaling, rotation, mirroring) - functions of the geometric moments. A normalization step: pattern \( f(x, y) \geq 0 \rightarrow \) pattern \( h(x', y') \geq 0 \),

global moments \( m_{pq} \rightarrow \mu_{pq} \)

\[ m_{pq} = \sum_x \sum_y x^p y^q f(x, y) \quad \mu_{pq} = \sum_x \sum_y x'^p y'^q h(x', y') \]

**Step 1:** shift to mass centre \( (x_c, y_c) \)

\[ x_c = \frac{m_{10}}{m_{00}}, \quad y_c = \frac{m_{01}}{m_{00}} \]

\[ x' = x - x_c, \quad y' = y - y_c \]

and normalize the amplitude

\[ h(x', y') = \frac{f(x, y)}{m_{00}} \]

\( \rightarrow \mu_{00} = 1 \) and \( \mu_{10} = \mu_{01} = 0 \)

Pattern normalization (2)

**Step 2:** size normalization – scaling of axes

\[ r = \sqrt{m_{02} + m_{20}} \]

\[ x' = \frac{x}{r}, \quad y' = \frac{y}{r} \]

\[ h(x', y') = r^2 f(x, y) \]

\( \rightarrow \mu_{20} + \mu_{02} = 1 \)

**Step 3:** rotation - to normalize the orientation of the pattern.

If \( m_{20} \neq m_{02} \) then minimize

\[ S(\alpha) = \sum_{(x, y) \in f} [(x - x_c) \cos \alpha - (y - y_c) \sin \alpha]^2 \]

and get:

\[ \tan(2\alpha) = \frac{2m_{11}}{m_{20} - m_{02}} \]
Pattern normalization (3)

Step 3 (cont.):
Among 4 possible angles select $\alpha$ such that after rotation it holds: $\mu_{20} < \mu_{02}$ and $\mu_{21} > 0$.

$$
\begin{pmatrix}
  x' \\
  y'
\end{pmatrix} =
\begin{pmatrix}
  \cos \alpha & \sin \alpha \\
  -\sin \alpha & \cos \alpha
\end{pmatrix}
\begin{pmatrix}
  x \\
  y
\end{pmatrix}
$$

$\rightarrow \mu_{11} = 0$, $\mu_{20} < \mu_{02}$ and $\mu_{21} > 0$

Step 4: mirroring with respect to the Y axis.

Select $\beta \in \{+1, -1\}$ such that after the transformation $x' = \beta x$, $y' = y$

the resulting pattern’s moment is $\rightarrow \mu_{12} > 0$

7. Image filters

Basic image filters are used to suppress:
• the high frequencies in the image, i.e. smoothing the image, or
• the low frequencies, i.e. detecting edges in the image.

Spatial filter: to convolve the input image $f(i,j)$ with some filter function $h(i,j)$ (called kernel):

$$
g(i, j) = h(i, j) * f(i, j)
$$

Frequency filter:
(1) transform the image into the frequency domain,
(2) multiply the result with the frequency filter function and
(3) re-transform the result into the spatial domain.

Discrete convolution is a „shift and multiply” operation: for a square kernel with size $(M+1) \times (M+1)$ the discrete convolution is:

$$
g(i, j) = \sum_{m=-M/2}^{M/2} \sum_{n=-M/2}^{M/2} h(m,n) \cdot f(i-m, j-n)
$$
Basic image filters:
1. **Mean filter** - noise reduction (NR) using mean of neighbourhood
2. **Median filter** - NR using median of neighbourhood
3. **Gaussian smoothing** - NR with a Gaussian smoothing kernel
4. Various **gradient-based** edge detection
5. **Laplace filter** - second derivation-based edge detection

**Mean filter:** to replace each pixel value in an image with the mean (average) value of its neighbours, including itself. The kernel:

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

**Median filter**

**Non-linear filter:** the addition operation in discrete convolution is replaced by some non-linear operator:

\[
g(i, j) = O_{m,n} [h(m,n) \cdot f(i - m, j - n)]
\]

**Median filter** is a non-linear filter. It replaces the pixel value with the median of neighbour values. The median is calculated by first sorting all the pixel values from the local neighbourhood according to numerical order and then replacing the central pixel by the middle value.

**Example:**

<table>
<thead>
<tr>
<th>123</th>
<th>125</th>
<th>126</th>
<th>130</th>
<th>140</th>
<th>3x3 neighbourhood values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>124</td>
<td>126</td>
<td>127</td>
<td>135</td>
<td>115, 119, 120, 123, 124,</td>
</tr>
<tr>
<td>118</td>
<td>120</td>
<td>150</td>
<td>125</td>
<td>134</td>
<td>125, 127, 127, 150</td>
</tr>
<tr>
<td>119</td>
<td>115</td>
<td>119</td>
<td>123</td>
<td>133</td>
<td>115, 119, 123, 133</td>
</tr>
<tr>
<td>111</td>
<td>116</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>Median value: 124.</td>
</tr>
</tbody>
</table>
Gaussian smoothing

Gaussian smoothing operator uses a kernel that approximates a Gaussian function. 2-D isotropic (circularly symmetric) Gaussian:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Discrete Gaussian kernel:
assume zero at distances more than three standard deviations from the mean and truncate the kernel.

Example: discrete approximation of Gaussian function with $\sigma = 1.0$.

The 2-D isotropic Gaussian is separable into $x$ and $y$ components → apply a 1-D convolution to rows and columns, with kernel:

W. Kasprzak: EIASR 4. Image pre-processing

Edge image detection

A pre-processing step in edge detection: a smoothing operation in order to remove noise (spiky-like variations) from the image.

Example: desired edge (left), real edge (right).

Basic types of edge image detectors:

1. discrete image function gradients,
2. convolve image with kernels,
3. using parametric edge models,
4. mixed approaches.

W. Kasprzak: EIASR 4. Image pre-processing
Discrete image gradients

The gradient of a 2-D continuous function:
\[
\nabla f(x, y) = \left( \begin{array}{c} f_x(x, y) \\ f_y(x, y) \end{array} \right) = \left( \begin{array}{c} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{array} \right)
\]

Discrete differences in case of a 2-D image \( \hat{f}(i, j) \) that represents the continuous function \( f(x, y) \):
\[
\hat{f}_x(i, j) = \hat{f}(i+1, j) - \hat{f}(i, j) \quad \hat{f}_y(i, j) = \hat{f}(i, j+1) - \hat{f}(i, j)
\]

As a result of edge detection two output images are computed:
1. the magnitude (strength) \( s \) or the "absolute" strength \( s' \) (for computational simplicity)
   \[
   s = \sqrt{f_x^2 + f_y^2} \quad s' = |f_x| + |f_y|
   \]
   and
2. the direction of the normal vector to edge:
   \[
   r = \arctan \left( \frac{f_y}{f_x} \right)
   \]

W. Kasprzak: EIASR 4. Image pre-processing

Robert’s cross

„Roberts cross”: example of a discrete gradient-based edge operator. Two discrete gradients along 45° and 135°:
\[
k_x(i, j) = f(i, j) - f(i+1, j+1), \quad k_y(i, j) = f(i+1, j) - f(i, j+1).
\]
Remark: they are equivalent to convolution kernels:
\[
\begin{array}{ccc}
+1 & 0 & 0 \\
0 & -1 & +1 \\
\end{array}
\]

Edge strength: \( s = \sqrt{k_x^2 + k_y^2} \) or \( s' = |k_x| + |k_y| \)

Edge orientation (correction by -3\(\pi/4\) needed):
\[
r_{Roberts} = \arctan \left( \frac{k_y}{k_x} \right) - \frac{3}{4} \pi
\]

Characteristics of the „Roberts Cross” operator.
- the same image format as input image, e.g. maximum strength: 255, for 8-bit images,
- simple implementation (two difference values per pixel only),
- sensitive to noise.
Differences of central element

Two differences that approximate the two gradients of image functions along the main axes:

\[ \hat{f}_x(i, j) = \hat{f}(i + 1, j) - \hat{f}(i - 1, j) \quad \hat{f}_y(i, j) = \hat{f}(i, j + 1) - \hat{f}(i, j - 1) \]

Remark: equivalent convolution kernels are:

\[
\begin{array}{ccc}
-1 & 0 & 1 \\
0 & 0 & 0 \\
1 & 0 & 1
\end{array}
\]

Characteristics:
- The same input-output format, e.g. maximum strength 255 for 1 byte/pixel),
- simple implementation,
- still sensitive to noise.

Convolution-based edge detector

**Sobel operator:**

\[
\begin{array}{ccc}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{array}
\]

\[
\begin{array}{ccc}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{array}
\]

**Prewitt operator:**

\[
\begin{array}{ccc}
-1 & 0 & 1 \\
-1 & 0 & 2 \\
-1 & 0 & 1
\end{array}
\]

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{array}
\]

Characteristics:
- Sobel operator: different input-output image, e.g. maximum strength: 2040 for 8-bit input image, simple implementation, good results.
- Prewitt operator: different input-output image, e.g. maximum strength: 1020 for 8-bit input image, simple implementation, results are almost as good as for the Sobel operator.
Discrete directions

A limited number of edge orientations is detected. This allows for an efficient implementation of orientation detection step.

For example 16 discrete image directions are detected by comparing two edge gradients for each pixel.

Laplace operator

Laplace operator (the sum of second-order derivatives)

For a continuous 2-D function the Laplace operator is defined as:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = f_{xx} + f_{yy}$$

The discrete Laplace operator uses a single mask but is independent from any direction (orientation information is lost).

It gives a „positive“ answer (i.e. a zero value) for both real edges and homogeneous regions in the image - it can be used in a combination with some other edge operator.

Basic discrete Laplace operator:

$$\nabla^2 f(i, j) = 4f(i, j) - f(i-1, j) - f(i+1, j) - f(i, j-1) - f(i, j+1)$$

The convolution kernel:

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>
Using a second derivative makes the Laplacian highly sensitive to noise → use Gaussian smoothing as a preprocessing step.

**Laplacian of the Gaussian (LOG filter)**

\[
L(x, y) = \nabla^2 (G(x, y) * I(x, y))
\]

Due to linearity:

\[
L(x, y) = \nabla^2 G(x, y) * I(x, y)
\]

Derivation of the LOG filter (1D case): \(G(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)\)

- first derivative:

\[
G'(x) = -\frac{x}{\sigma^3 \sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)
\]

- second derivative: \(G''(x) = \frac{1}{\sigma^5 \sqrt{2\pi}} \left(\frac{x^2 - 1}{\sigma^3} \right) \exp\left(-\frac{x^2}{2\sigma^2}\right)\)

LOG filter (2)

Illustration of the LOG filter:

Different Laplace convolution kernels:

(a) Basic Laplacian

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

(b),(c) LOG filter

(a) \[
\begin{pmatrix}
0 & -1 & 0 \\
-1 & 8 & -1 \\
0 & -1 & 0 \\
\end{pmatrix}
\]

(b) \[
\begin{pmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-2 & 4 & -2 \\
\end{pmatrix}
\]

(c) \[
\begin{pmatrix}
1 & -2 & 1 \\
1 & -2 & 1 \\
-1 & -2 & 1 \\
\end{pmatrix}
\]
Color image edge operator

An edge operator defined for a monochromatic image can be extended to handle color image.

For example let be an RGB image function \( f_{RGB} \).

Let pairs of colour pixels be given: \( f_1 = (r_1, g_1, b_1) \), \( f_2 = (r_2, g_2, b_2) \) - from the local neighbourhood of current pixel.

In case of the Sobel color operator there will be given three pairs for the "vertical" mask and 3 pairs for the "horizontal" mask.

The difference for pair \((f_1, f_2)\) can be alternatively computed as:

\[
D_1 (f_1 ; f_2) = \{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2\}^{1/2}
\]

\[
D_2 (f_1, f_2) = |r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2|
\]

\[
D_3 (f_1, f_2) = \max \{ |r_1 - r_2|, |g_1 - g_2|, |b_1 - b_2| \}
\]

\[
D_4 (f_1, f_2) = \omega_r |r_1 - r_2| + \omega_g |g_1 - g_2| + \omega_b |b_1 - b_2|
\]

8. Edge thinning

Threshold-based edge elimination

This simple edge thinning method is an edge elimination operator with a minimum threshold parameter \( \theta \). The threshold is either fixed or set adaptively (e.g. \( \theta = \gamma S_{\text{max}} \), where \( \gamma \in (0,1) \)).

\[
s_{\text{thin}}(P) = \begin{cases} s(P), & \text{if } s(P) > \theta \\ 0, & \text{otherwise} \end{cases}
\]

No-maximum edge elimination

It depends on a check in the local neighborhood of given pixel \( P \)

\[
\text{IF} \ ((s(P) \geq s(N_L) \ \text{OR} \ |r(P) - r(N_L)| \geq T) \\
\text{AND} \ (s(P) \geq s(N_R) \ \text{OR} \ |r(P) - r(N_R)| \geq T)) \\
\text{THEN} \ s(P)_{\text{thin}} = s(P); \ \text{ELSE} \ s(P)_{\text{thin}} = 0;
\]
Edge thinning (2)

Edge modification

A **local neighborhood-based modification** of edge at pixel $P$:

- if $P$ is the strongest edge element in the set: $P, N_L, N_R$, then:
  \[ s'(P) = s(P) + \alpha (s(N_L) + s(N_R)) \]
- if $P$ is the weakest edge element in the above set then:
  \[ s'(P) = s(P) - 2\alpha s(P) \]
- if one neighbour of $P$ (denoted by $P^+$) is a stronger edge and another neighbour of $P$ (denoted by $P^-$) is a weaker edge element then:
  \[ s'(P) = s(P) - \alpha s(P^+) + \alpha s(P^-) \]

Several iterations over the whole image may be necessary.

---

Edge thinning (3)

**Edge elimination with hysteresis threshold**

This edge thinning method works with two edge strength thresholds: the upper $\theta_H$ and the lower $\theta_L$.

In the first run these edge pixels are individually marked as „good” that have higher strengths than the upper threshold.

In the next run these “good” pixels are tried to be “extended” along a potential contour line in both directions (“positive” and “negative” line direction). For a single extension, the neighbor pixel need to have higher strength than the lower threshold.

**Remark:** Now the neighbors are not competing with each other and they are searched along the expected contour line (not across it).
9. Canny operator

This a multi-stage edge image detection and thinning procedure. It needs 3 parameters:

• the variance $\sigma$ for the Gaussian mask,
• edge strength thresholds $\theta_H$, $\theta_L$, where $\theta_H > \theta_L$, for hysteresis-based thinning.

**Input**: a grey-scale image. **Output**: a “thinned” edge image.

**Steps of the Canny operator**

1. Image smoothing by convolution with a Gaussian kernel.
2. Edge image detection by a discrete-gradient operator in 2x2 or 3x3 neighbourhood.
3. Edge thinning by the no-maximum elimination operator.
4. Edge elimination with a hysteresis threshold.

W. Kasprzak: EIASR 4. Image pre-processing
1. Line detection

Main approaches to line detection in images:
1. A 3-step line segment detection
   • Edge image detection (filtering),
   • Edge chain following (segmentation)
   • To approximate edge chains by straight line segments (symbolic description).
2. A 2-step line or contour detection
   • Edge image detection
   • Detection of straight lines, circles and contours by using a Hough transform.
   • „Active contour” method

2. Edge chain following

Principle: searching for extension of current edge pixel $P=\text{P}_{\text{cur}}$ by its successor edge $N=\text{c} (\text{P}_{\text{cur}})$.

Two neighbour edge elements can be linked if the edge magnitude (strength) and direction differences are below certain thresholds and their magnitudes are relatively large:

\[
|s(p) - s(n)| \leq T_1
\]
\[
|r(p) - r(n)| \mod 2\pi \leq T_2
\]
\[
|s(p)| > T, \quad |s(n)| > T
\]

Denote the 3 nearest pixel along the direction $r(p)$ as: $N_1, N_2, N_3$.

The successor of $\text{P}_{\text{cur}}$: an edge element $N_i$ whose strength and orientation are most similar to $\text{P}_{\text{cur}}$. 
Successor candidates

Three candidates for a successor of pixel P:

Closing a gap for edge pixel P in directions (left) $0^\circ$ and (right) $45^\circ$:

The hysteresis threshold method

The edge strength threshold $T$ is now split into two:
- the upper threshold $T_0$ and
- the lower threshold $T = \beta T_0$, $0 < \beta < 1$

The strength of the start edge pixel has to be higher than the upper threshold $T_0$: $|s(P_{\text{start}})| > T_0$

The strength of any next edge element in the chain needs to be higher than $T$.

Example.

Edge chain labels:
The hysteresis threshold (2)

REPEAT
Search for an edge pixel \( P_{\text{start}} \) with no segment label (i.e. „free”) with strength \( s(P_{\text{start}}) > T_o \) (upper threshold) .

\( P_{\text{start}} \) is now called current pixel \( P_{\text{cur}} \) and the segment label is \( \text{SegNum} \).

FORWARD_SEARCH (\( P_{\text{cur}} \))

IF chain \( \text{SegNum} \) is not closed

THEN Again set the current pixel \( P_{\text{cur}} \) to be the start pixel \( P_{\text{start}} \).

Perform BACKWARD_SEARCH(\( P_{\text{cur}} \))

UNTIL all pixels are visited.

Example:

<table>
<thead>
<tr>
<th>Y</th>
<th>X</th>
<th>1 (s, r)</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21, 8</td>
<td>22, 8</td>
<td>11, 10</td>
<td>6, 12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30, 8</td>
<td>34, 8</td>
<td>35, 9</td>
<td>20, 10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6, 11</td>
<td>37, 8</td>
<td>36, 9</td>
<td>33, 10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>13, 14</td>
<td>12, 15</td>
<td>20, 8</td>
<td>30, 10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12, 13</td>
<td>12, 10</td>
<td>11, 7</td>
<td>12, 8</td>
<td></td>
</tr>
</tbody>
</table>

W. Kasprzak: EIASR 5. Boundary-based image segmentation

Edge chain image

Input image
3. Line segment fit

The equation of a line determined by two points \((x_j, y_j), (x_k, y_k)\):

\[ x(x_i - y_k) + y(x_k - x_j) = y_j(x_k - x_j) - x_j(y_k - y_j) \]

\[ a x + b y = c, \quad a^2 + b^2 > 0 \]

The distance \(d_i\) of point \((x_i, y_i)\) from this line:

\[ d_i = \frac{|ax_i + by_i - c|}{\sqrt{a^2 + b^2}} \]

---

Edge chain approximation

**Procedure APPROXIMATE** (CI, P1, PN)

Add the line \(l\) determined by start point P1 and end point PN of the chain CI to the set of line segments \(L(C)\).

For every point \(P_k\) in chain CI determine the distance \(s_k\) to line \(l\).

IF FOR ALL \(P_k \in CI: s_k \leq \theta\) THEN RETURN (END).

Select a point \(P_k\) with maximum distance \(s_k\).

Remove the line \(l\) from set \(L(C)\).

Decompose the line CI into 2 parts

\(CI_1 = P_1 \ldots P_k; CI_2 = P_k \ldots P_N\).

Call APPROXIMATE(CI_1, P_1, P_k).

Call APPROXIMATE(CI_2, P_k, P_N).
Approximation by arcs (1)

1. Try to approximate two consecutive line segments by an arc. E.g. start with arc ACB.

2. Measure the approximation error

E.g. 
\[ e_A = \arg \max_i |g(x_i) - f(x_i)|, \quad i = 1,2,3; \]
\[ e_B = \arg \max_i |g(x_i) - f(x_i)|, \quad i = 5,6,7; \]

if \( \text{sign}(g(x_{eA}) - f(x_{eA})) = \text{sign}(g(x_{eB}) - f(x_{eB})) \)
AND \( \max(|g(x_{eA}) - f(x_{eA})|, |g(x_{eB}) - f(x_{eB})|) < \gamma R \)
THEN Approximate(ACB, R)

Approximation by arcs (2)

Approximate(ACB, R)
Approximation by arcs (3)

3. Try to extend an existing arc by including the next line segment
E.g., arc ACB is checked to be extended to ACBD. Then an extension to
ACBDF fails, but there is a second arc DFE.

4. Try to approximate closed arc chains by a circle or ellipses.

4. Hough transform (HT)

A polar representation of a straight line defined by an edge element \( e_i \) consists of two parameters: \( d, \alpha \), where

\[
d = x_i \cos \alpha + y_i \sin \alpha
\]

Remark: \( \alpha = 90^\circ + \beta \), and \( \alpha = r(e) \) is the edge’s direction.

An image line corresponds to a point in the Hough space \( (d; \alpha) \).
Hough transform for lines

**Edge image** \( E = [ e_{ij} ] \), \( i = 1, \ldots, M; \ j = 1, \ldots, N; \) is of size \( M \times N \) and it represents \( O_{\text{NUM}} \) discrete edge directions.

The **Hough accumulator** \( H(d; \alpha) \) is a digital representation of the Hough space:

\[
- \frac{1}{2} \sqrt{M^2 + N^2} \leq d \leq \frac{1}{2} \sqrt{M^2 + N^2}, \quad -\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}
\]

**Hough Transform for line detection**

FOR every edge pixel \( e_{ij} \) (coordinates \((x_i, y_i)\), orientation \( \alpha_{ij} \)) DO:
- Compute \( d_{ij} = x_i \cos \alpha_{ij} + y_i \sin \alpha_{ij} \);
- Approximate \((d_{ij}, \alpha_{ij})\) by nearest discrete values \((d_{ij}^+, \alpha_{ij}^+)\) and increase \( H(d_{ij}^+, \alpha_{ij}^+) \), i.e. \( H(d_{ij}^+, \alpha_{ij}^+) = H(d_{ij}, \alpha_{ij}) + 1 \).

FOR every \((d, \alpha)\) DO:
- IF \( H(d, \alpha) > \text{Threshold} \) THEN corresponding line is detected.

HT for circle center detection

A circle, \((x - x_c)^2 + (y - y_c)^2 = r^2\), has 3 parameters.

**A two-step circle recognition approach**

**Step 1:** detect circle centres by appropriate Hough transform;

**Step 2:** detect edge chains around those centres.

**Hough Transform for circle center detection**

Let a line perpendicular to edge element \( e_i \) be: \( g_i: y = a_i x + b_i \)

Lines corresponding to circle border points cross in the point \( C = (x_c, y_c) \) - the centre point of circle.

In Hough space \( H(a,b) \) these lines will correspond to collinear points \((a_i, b_i)\).
Circle center detection

A line \( g_i \) that passes through \((x_c, y_c)\) satisfies:

\[
y = a_i(x - x_c) + y_c = a_i x + (y_c - a_i x_c).
\]

This line corresponds to the point in Hough space: \((a_i; y_c - a_i x_c)\).

The relation of a set of points in Hough space is approximately a straight line:

\[
b_i = -a_i x_c + y_c.
\]

For \(N\) point observations:

\[
\begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_N
\end{bmatrix} =
\begin{bmatrix}
  -a_1 & 1 \\
  -a_2 & 1 \\
  \vdots & 1 \\
  -a_N & 1
\end{bmatrix}
\begin{bmatrix}
  x_c \\
  y_c
\end{bmatrix}
\]

the LSE solution is:

\[
\begin{bmatrix}
  -x_c \\
  y_c
\end{bmatrix} = \frac{1}{N \sum a_i^2 - (\sum a_i)^2} \begin{bmatrix}
  N \\
  - \sum a_i \\
  \sum a_i^2 \\
  \sum b_i
\end{bmatrix} \begin{bmatrix}
  \sum (a_i b_i)
\end{bmatrix}
\]

HT for circle detection

Now let us apply the Hough transform to detect all 3 parameters of a circle equation, \((x - x_c)^2 + (y - y_c)^2 = r^2\), in one step.

**Hough transform for circle detection**

FOR every edge pixel \(e_i\) (coordinates \((x_i, y_i)\), orientation \(\alpha_i\)) DO:
- the (unknown) centre coordinates \((x_C, y_C)\) of a circle with (unknown) radius \(r_d\) passing through point \((x_i, y_i)\) satisfy:
  \[
x_i = x_C + r_d \cos \alpha_i, \quad y_i = y_C + r_d \sin \alpha_i;
\]
- FOR every discrete radius \(r_d\): \(0 \leq r_d \leq r_{\text{max}}\), DO:
  - compute \((x_C, y_C)\) from the above equations and
  - increase the Hough accumulator \(H(x_C, y_C, r_d)\) by one;

FOR every \((x_C, y_C, r_d)\) DO:
- IF \(H(x_C, y_C, r_d) > \text{Threshold}\) THEN corresponding circle is detected.
Contour detection by GHT

A generalised Hough transform (GHT) for contour detection

Parameters of the Hough space:

\[ C = (x_C; y_C) \]: location of the center mass,
\[ s \]: scale,
\[ \alpha \]: contour orientation angle.

Model learning (design)

For every pair of allowed discrete values \( s_d, \alpha_g \) of scale and orientation, create a table \( R(s_d, \alpha_g) = [r(\phi), \phi] \), where for each edge \( x_B \) with edge direction \( \phi \) the pair: \( [r(\phi) = x_B - C, \phi] \) is stored.

Example: The model contour (top drawing) and the candidate contour (bottom)

---

5. Active contour

The active contour, or snake, is defined as an energy minimizing polygon - the snake’s energy depends on its shape and location within the image.

A snake is initialized roughly to represent the object’s boundary. The snake’s motion rule is then iteratively applied to change the snake’s location and shape to satisfy the final condition:

\[
F_{\text{internal}} + F_{\text{external}} = 0
\]

\[
F_{\text{external}} = -\nabla E_{\text{external}}
\]

Example.

Two final snakes established from common initial snake while using more (outer) or less (inner) external force.
Snake’s forces

Edge-based external energy:

\[ E_{\text{external}}^{(1)}(x, y) = -|\nabla I(x, y)|^2 \]

or

\[ E_{\text{external}}^{(2)}(x, y) = -|\nabla (G_\sigma(x, y) * I(x, y))|^2 \]

The internal energy is responsible for elasticity and stiffness – trying to shorten and to smooth the contour:

\[ E_{\text{internal}} = E_{\text{elastic}} + E_{\text{stiffness}} \]

E.g.

\[ E_{\text{elastic}} = K_1 \cdot \sum_{i=0}^{n-1} |p_i - p_{i-1}|^2 \]

\[ E_{\text{stiffness}} = K_2 \cdot \sum_{i=0}^{n-1} |p_{i-1} - 2p_i + p_{i+1}|^2 \]

W. Kasprzak: EIASR 5. Boundary-based image segmentation

Motion (update) rule

Snake dynamics – discrete motion rule: at each iteration, every control point \( p_i = (x_i, y_i) \) is moved by a vector proportional to the force acting on it: 

\[ x_i = x_i + \alpha F_{\text{elasticX},i} + \beta F_{\text{stiffnessX},i} + \gamma F_{\text{externalX},i} \]

\[ y_i = y_i + \alpha F_{\text{elasticY},i} + \beta F_{\text{stiffnessY},i} + \gamma F_{\text{externalY},i} \]

For example: \( \alpha = 0.8, \beta = 0.1, \gamma = 0.6 \)

In particular:

\[ F_{\text{elasticX},i} = 2K_1 \left( (x_{i-1} - x_i) + (x_{i+1} - x_i) \right) \]

\[ F_{\text{elasticY},i} = 2K_1 \left( (y_{i-1} - y_i) + (y_{i+1} - y_i) \right) \]

\[ F_{\text{stiffnessX},i} = K_2 \left( 4(x_{i-1} + x_{i+1}) - 6x_i - (x_{i-2} + x_{i+2}) \right) \]

\[ F_{\text{stiffnessY},i} = K_2 \left( 4(y_{i-1} + y_{i+1}) - 6y_i - (y_{i-2} + y_{i+2}) \right) \]

\[ F_{\text{externalX},i} = \frac{K_3}{2} \left( I(x_i + 1,y_i) - I(x_i - 1,y_i) \right) \]

\[ F_{\text{externalY},i} = \frac{K_3}{2} \left( I(x_i,y_i + 1) - I(x_i,y_i - 1) \right) \]
Example

Two different final contours – obtained by varying the weights of the internal force.

Inner contour:
\[ \alpha = 0.8, \beta = 0.5, \gamma = 0.6 \]

Outer contour:
\[ \alpha = 0.8, \beta = 0.1, \gamma = 0.6 \]

6. Point detector

The Harris-Stephens operator

Detect average image gradients \( I_x, I_y \) in the local neighborhood of image point \((x_k, y_k)\) – represent them as a covariance matrix:

\[
A(x_k, y_k) = \begin{bmatrix}
\sum_{W} (I_x(x_k, y_k))^2 & \sum_{W} I_x(x_k, y_k) I_y(x_k, y_k) \\
\sum_{W} I_x(x_k, y_k) I_y(x_k, y_k) & \sum_{W} (I_y(x_k, y_k))^2
\end{bmatrix}
\]

A point feature is detected when both eigenvalues of matrix \( A \) are of high and similar value.
SIFT and SURF detector


SIFT implementation consists of following steps:
1. Scale-and-image space filtering (LoG filter)
2. Detection of extrema in scale-image space (DoG filter)
3. Keypoint (corner feature) detection,
4. Dominating orientation “at keypoint” is detected,
5. Keypoint description.

**SURF** (*Speeded Up Robust Features*) [Bay et al., 2008]

Idea: DoH - *Determinant of Hessian* for keypoint detection.
1. Homogeneous region

A region $R$ is a connected image area, which is homogeneous with respect to some parameter (vector of parameters) (e.g. intensity, colour, texture), and given predicate, $H(R) = 1$.

Examples of homogeneity predicate

1. $H(R) = \begin{cases} 1, & \text{if } |\max(R) - \min(R)| < \theta(R) \\ 0, & \text{otherwise} \end{cases}$

where: $\max(R) = \max_{p \in R} f(p)$, $\min(R) = \min_{p \in R} f(p)$

2. $H(p, R) = \begin{cases} 1, & \text{if } |f(p) - \text{mean}(R)| < \theta(R) \\ 0, & \text{otherwise} \end{cases}$

where: $\theta(R) = \theta_A - \frac{F(R)}{F(R_{\max})} (\theta_A - \theta_B)$

or

$\theta(R) = k\sigma = k \sqrt{\frac{1}{n} \sum_{i=1}^{n} (f(p_i) - \text{mean}(R))^2}$; where $p_i \in R, i = 1, \ldots, n$.

Region growing and merging

A simple approach to region detection is to start from some pixels (called seeds) $s_i$ representing distinct image regions $R_i$ and to grow them, until they cover the entire image.

The initial seed detection: we make a histogram of the entire image; image pixels, whose image value corresponds to histogram peaks, are selected to be seeds.

**Region growing**: at each stage $k$ and for each region $R_i^{(k)}$ check if there are unlabeled neighbour pixel of each pixel of the region border and if the homogeneity criterion for the enlarged region will still be valid. If it is true then enlarge the region by this pixel.
Split-and-merge

Region merging: merge adjacent regions that have similar statistical properties. For example the two regions $R_1$, $R_2$ are allowed for merge if their arithmetic means are similar:

\[ \text{Predicate (3)} \quad |\text{mean}(R_1) - \text{mean}(R_2)| < \theta(R_1 \cup R_2) \]

Split-and-merge (quadtree structure)

Init: quadtree at some mid-level

SPLIT: check whether every leaf node is homogeneous

MERGE: check whether 4 related leafs can be merged

Combined region detection algorithm $(M, \theta, \theta_A, \theta(R))$:

Initial image tessellated into square regions of size $M \times M$.

REPEAT with the quadtree representation DO

- MERGE step (with a fixed threshold $\theta$)
- SPLIT step (with a fixed threshold $\theta$)

UNTIL further split or merge is not possible (predicate (1)).

Region merging/growing with a fixed threshold $\theta_A$ (pred. (3))

Region merging with an adaptive threshold $\theta(R)$ (pred. (2))
2. Texture

**Texture** - a measure of image regularity, coarseness and smoothness.

Examples of textures: grass, wood, water surface, etc.

Texture classification

**Texture recognition** is a **classification** problem.

**Learning:** find appropriate features, train a classifier,

**Classification:** detect features of current texture; classify the features in terms of learned classes.

**Texture features:**

- **statistical** (e.g. variance, skewness);
- **spectral** (e.g. using the autocorrelation function or Fourier transform);
- **structural** (e.g. using discrete features like: colour, motion, number of endings).
Histogram-based texture features

Let \( f_k, k=1,\ldots,L \), be the values of the image function, and \( p(f_k) \) the normalized histogram of an image region.

1. Mean
   \[
   \mu_1 = \mu = \sum_{k=1}^{L} f_k p(f_k)
   \]

2. Variance
   \[
   \mu_2 = \sigma^2 = \sum_{k=1}^{L} (f_k - \mu_1)^2 p(f_k)
   \]

3. Skewness
   \[
   \mu_3 = \frac{1}{\sigma^3} \sum_{k=1}^{L} (f_k - \mu_1)^3 p(f_k)
   \]

4. Kurtosis
   \[
   \mu_4 = \frac{1}{4} \sum_{k=1}^{L} (f_k - \mu_1)^4 p(f_k) - 3
   \]

5. Entropy
   \[
   H(f) = -\sum_{k=1}^{L} p(f_k) \cdot \log p(f_k)
   \]

The interpretation of features

1. The **mean** gives the average pixel value in given region
2. The **variance** measures the dispersion of pixel values in this region.
3. **Skewness** is a measure of histogram symmetry.
4. **Kurtosis** is a measure of the tail of the histogram – long-tailed histograms correspond to spiky regions. The subtraction of 3 ensures that the kurtosis of a Gaussian distribution is normalized to zero.
5. **Entropy** expresses the level of uniform pixel value distribution.

The major limitation of histogram features is that they cannot express spatial characteristics of the texture.
Histograms of pairs

Histograms of sums and differences

Let $f_{j,k}$, $f_{j+\mu,k+v}$ be two pixel in the image separated by the displacement vector: $V = (\mu; \nu)^T$. Their sum and difference are:

$$s_{j;k} = f_{j;k} + f_{j+\mu,k+v}; \quad d_{j;k} = f_{j;k} - f_{j+\mu,k+v}$$

Histograms of sums and differences, $H_s(l,V)$ and $H_d(m,V)$, for displacement $V$:

$$H_s(l,V) = \#f_{j;k} \text{ (such that } s_{j;k} = l)$$
$$H_d(m,V) = \#f_{j;k} \text{ (such that } d_{j;k} = m)$$

They contain information about the spatial organization of pixel values. E.g. a coarse texture results in a concentration of histograms around: $l = 2 \times \text{mean}$ and $m=0$.

Consider several displacements, e.g.:

$$V = (\mu; \nu)^T \in \{(1,0), (0,1), (-1,0), (0,-1)\}$$

Features of sums and differences

After normalisation:

$$h_s(l) = \frac{H_s(l)}{\sum_l H_s(l)}, \quad l = 0,\ldots,2L-1 \quad h_d(m) = \frac{H_d(m)}{\sum_m H_d(m)}, \quad m = (-L+1),\ldots,(L-1)$$

Typical features are:

1. Mean

$$c_1 = \sum_{l=0}^{2L-2} lh_s(l) \quad c_2 = \sum_{m=-L+1}^{L-1} mh_d(m)$$

2. Contrast

$$c_3 = \sum_{l=0}^{2L-2} (l^2 \cdot h_s(l)) \quad c_4 = \sum_{m=-L+1}^{L-1} (m^2 \cdot h_d(m))$$

3. Variance

$$c_5 = \sum_{l=0}^{2L-2} (l - c_1)^2 h_s(l) \quad c_6 = \sum_{m=-L+1}^{L-1} (m - c_2)^2 h_d(m)$$

4. Entropy

$$c_7 = \sum_{l=0}^{2L-2} -h_s(l)\log(h_s(l)) \quad c_8 = \sum_{m=-L+1}^{L-1} -h_d(m)\log(h_d(m))$$
The grey-level co-occurrence matrix (GLCM) (by R. Haralick et al.) is defined as: 

\[ G(d; r) = [g_{ij}(d; r)] \]

where an element \( g_{ij}(d; r) \) specifies the number of pixel pairs \((f_i = i, f_j = j)\) which are separated by distance \( d \) along the direction \( r \).

**Example.** For a 5 × 5 image and \( L (L = 4) \) values, the co-occurrence matrix has \( L^2 = 4 \times 4 \) elements.

\[
f \rightarrow \begin{pmatrix}
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 2 & 2 & 2 & 0 \\
2 & 2 & 3 & 3 & 0 \\
2 & 2 & 2 & 2 & 2
\end{pmatrix}
\]

\[ G(1; 0) = \begin{pmatrix}
4 & 4 & 2 & 1 \\
4 & 4 & 0 & 0 \\
2 & 0 & 14 & 1 \\
1 & 0 & 1 & 2
\end{pmatrix} \]

E.g.

\[ g_{00}(1,0) \rightarrow (f_{00}, f_{01}), (f_{01}, f_{00}), (f_{10}, f_{11}), (f_{11}, f_{10}) \]

Parameters of GLCM

The adjacency of pixels in a pair – along four directions \( r \) (horizontal, vertical, left and right diagonal):

The distance parameter \( d \), is usually set as \( d = 1 \), but it can be set as \( d > 1 \).

R. Haralick proposed fourteen features for the GLCM.
Features of the GLCM

Normalisation of GLCM: 
\[ \overline{g}_{ij}(d,r) = \frac{g_{ij}(d,r)}{\sum_{i,j} g_{ij}(d,r)}, \quad i, j = 0, \ldots, (L-1) \]

Marginal distributions: 
\[ g_{1i} = \sum_{j=0}^{L-1} \overline{g}_{ij} \quad g_{2j} = \sum_{i=0}^{L-1} \overline{g}_{ij} \]

Mean and standard deviation of marginal distribution:
\[ \mu_1 = \sum_{i=0}^{L-1} i \cdot g_{1i} \quad \sigma_1 = \sqrt{\sum_{i=0}^{L-1} (i - \mu_1)^2 \cdot g_{1i}} \]
\[ \mu_2 = \sum_{j=0}^{L-1} j \cdot g_{2j} \quad \sigma_2 = \sqrt{\sum_{j=0}^{L-1} (j - \mu_2)^2 \cdot g_{2j}} \]

Selected features:
1. Mean-square of distribution: 
\[ c_1 = \sum_{i,j} \overline{g}_{ij}^2 \]
2. Contrast of the „difference”-variable: 
\[ c_2 = \sum_{l=0}^{L-1} l^2 \cdot (\sum_{|i-j|=l} \overline{g}_{ij}) \]
3. Normalized “correlation” of marginal distributions: 
\[ c_3 = \frac{\sum_{i,j} (i - \mu_1)(j - \mu_2) \cdot \overline{g}_{ij}}{\sigma_1 \cdot \sigma_2} \]
4. Entropy: 
\[ c_4 = -\sum_{i,j} \overline{g}_{ij} \cdot \log \overline{g}_{ij} \]
Filter banks

**Convoluting** an image block with a set of kernels yields a representation of block’s texture in a different space. There is a **strong response** when the texture looks similar to the filter kernel, and a weak response when it doesn’t.

**Example.** The collection of kernels may consist of a series of **spots** and **bars** - at different **scales**.

**Spot** filters respond strongly to small (non-oriented) points. **Bar** filters are oriented, and tend to respond to edges.

---

Gabor filter banks

**Gabor filters** - Fourier Transform elements **multiplied** by Gaussians.

A Gabor kernel responds strongly if located over image blocks with texture having a particular **spatial frequency** and **orientation**. Gabor filters come in **pairs**: symmetric and anti-symmetric.

A symmetric kernel:

\[
G_{\text{symmetric}}(x, y) = \cos(k_x x + k_y y) \exp\left\{-\frac{x^2 + y^2}{2\sigma^2}\right\}
\]

An anti-symmetric kernel:

\[
G_{\text{anti-symmetric}}(x, y) = \sin(k_x x + k_y y) \exp\left\{-\frac{x^2 + y^2}{2\sigma^2}\right\}
\]

Where \((k_x, k_y)\) are the spatial frequencies.
**Example**

**Example.** Gabor filter kernels, where mid-grey values represent a zero, darker values represent negative numbers and lighter values represent positive numbers. The top row shows three anti-symmetric kernels, and the bottom row - three symmetric kernels; in horizontal direction.

Gabor filters in MPEG-7

**MPEG-7 descriptors in the frequency domain:** in polar coordinates the frequency domain is sampled into 30 "property channels" – the phase’s width is uniformly of 30 degree, and the magnitude widths are powers of 2:

\[ \theta_r = 30^\circ r , \ (r = 0,1,\ldots,5) \quad \omega_s = \omega_0 \cdot 2^{-s} , \ (s = 0,1,\ldots,4), \ \omega_0 = \frac{3}{4} \]
Gabor filters in MPEG-7 (cont.)

The Gauss function in channel \((s, r)\):

\[
G_{ps,r}(\omega, \theta) = \exp\left[\frac{-\left(\omega - \omega_s\right)^2}{2\sigma^2_{\rho_s}}\right] \cdot \exp\left[\frac{-\left(\theta - \theta_s\right)^2}{2\sigma^2_{\theta_s}}\right]
\]

The energy \(e_i\) of the \(i\)-th channel:

\[
e_i = \log\left(1 + g_i\right), \quad (i = 1, 2, \ldots, 30)
\]

\[
g_i = \sum_{\omega=0^\circ}^{180^\circ} \sum_{\theta=0^\circ}^{180^\circ} |G_{ps,r}(\omega, \theta) \cdot F(\omega, \theta)|^2
\]

The standard deviation of energy \(q_i\) in the \(i\)-th channel:

\[
q_i = \log\left(1 + \sigma_i\right), \quad (i = 1, 2, \ldots, 30)
\]

\[
\sigma_i = \sqrt{\sum_{\omega=0^\circ}^{180^\circ} \sum_{\theta=0^\circ}^{180^\circ} \left|G_{ps,r}(\omega, \theta) \cdot F(\omega, \theta)\right|^2 - g_i / N}
\]

Where \(F(\omega, \theta)\) – 2D Fourier transform into the frequency space represented by polar coordinates.

3. Point-based image descriptor

The image descriptor in SIFT

**Dominating orientation.** An orientation is assigned to each keypoint to achieve invariance to image rotation. A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. Eventually, this replicates the keypoint into many, with same location and scale, but different directions.

**SIFT feature descriptor.**

A 16s x 16s neighbourhood around the keypoint is taken. It is divided into 16 sub-blocks of 4 x 4 size. For each sub-block, an 8-bin orientation histogram is created. Thus, 128 bin values are available. The dominating direction is used for registration of the feature vector.
SURF descriptor

**Dominating orientation.** SURF uses wavelet responses in horizontal and vertical direction for a neighborhood of size 6s, with adequate Gaussian weights, to obtain gradients $dx$, $dy$. The dominant orientation is estimated by calculating the sum of all responses in the $(dx, dy)$ space, contained within a sliding orientation window of angle 60 degrees.

**SURF feature descriptor.**
A neighborhood of size $20s \times 20s$ is taken around the keypoint, where $s$ is the scale. It is divided into $4 \times 4$ subregions. For each subregion, horizontal and vertical wavelet responses are taken, and a vector is formed:

$$v = [\sum dx, \sum dy, \sum |dx|, \sum |dy|]$$

This gives a SURF feature descriptor with total 64 dimensions.

An extended descriptor with 128 elements is also defined: the sums of $dx$ and $|dx|$ are computed separately for $dy < 0$ and $dy \geq 0$; similarly, the sums of $dy$ and $|dy|$ are computed separately according to the sign of $dx$.

---

4. 2-D shape features

**Nominal numbers.** Distances from two base lines (lower and upper), measured at selected points. The numbers of crossing points of the contour and a given set of lines.

**Histograms.** Projecting the pattern onto its coordinate system axes results in two 1-D distributions (histograms).
Geometry features

Compactness features

• Area $A$, boundary length $L$
• Compactness coefficient: $\gamma = \frac{L^2}{A}$
  
  e.g. $\gamma = 12.6$ (for a circle), 16 (for a square)

  \[
  \frac{L^2}{A} = 4\pi = 12.6 \quad \frac{L^2}{A} = 16 \quad \frac{L^2}{A} = 18
  \]

• Normalized compactness: $\gamma_N = 1 - \frac{4\pi A}{L^2}$
  
  e.g. $\gamma_N = 0$ for circles and $\gamma_N \to \infty$ for complex shapes.

Moment-based features

Global and central geometric moments of the 2-D shape:

\[
m_{pq} = \sum_x \sum_y x^p y^q \cdot p(x, y) \quad \mu_{pq} = \sum_x \sum_y (x-x_c)^p (y-y_c)^q \cdot p(x, y)
\]

1) All central moments are translation-invariant.

2) The following moment-based functions \{\phi_1, \ldots, \phi_7\} are translation- and rotation-invariant:

\[
\begin{align*}
\phi_1 &= \mu_{20} + \mu_{02} \\
\phi_2 &= (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \\
\phi_3 &= (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2 \\
\phi_4 &= (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2 \\
\phi_5 &= (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + \\
&\quad + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] \\
\phi_6 &= (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03}) \\
\phi_7 &= (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + \\
&\quad - (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]
\end{align*}
\]
Moment-based features (2)

The moment-functions \( \{ \phi_1, \ldots, \phi_6 \} \) are reflection- (mirroring-) invariant as well. In case of \( \phi_7 \) its magnitude is reflection invariant but its sign changes under reflection.

The moment-functions \( \{ \phi_2, \ldots, \phi_7 \} \) can be made scale-invariant after a normalization by:

\[
    r = \sqrt{\mu_{20} + \mu_{02}}
\]

\[
    \phi_2' = \frac{\phi_2}{r^4}, \quad \phi_3' = \frac{\phi_3}{r^6}, \quad \phi_4' = \frac{\phi_4}{r^6}
\]

\[
    \phi_5' = \frac{\phi_5}{r^{12}}, \quad \phi_6' = \frac{\phi_6}{r^8}, \quad \phi_7' = \frac{\phi_7}{r^{12}}
\]

Example

Different shapes distinguished by using moment-based features.
5. Contour features

Let the contour polygon that approximates a 2D shape be mapped to a 1-D signal.

For example, the amplitude values correspond to segment angle changes, sampled at (cumulative length) segment’s end points. Features correspond to DFT (Fourier) coefficients of the 1D signal.

Discrete Fourier Transform

Let be: \( x = [x_0, x_1, \ldots, x_{M-1}] \) – a digital 1D signal.

The Discrete Fourier Transform (DFT) of order M is:

\[
F_k = \sum_{t=0}^{M-1} x_t \cdot e^{-i 2\pi k t / M}, \quad k = 0, \ldots, M - 1
\]

In matrix form:

\[
F = D_M \cdot x
\]

\[
\begin{bmatrix}
F_0 \\
F_1 \\
F_2 \\
\vdots \\
F_{M-1}
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & m & m^2 & \ldots & m^{M-1} \\
1 & m^2 & m^4 & \ldots & m^{2(M-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & m^{M-1} & m^{2(M-1)} & \ldots & m^{(M-1)^2}
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
\vdots \\
x_{M-1}
\end{bmatrix}
\]

\[
m = e^{-i 2\pi / M}
\]
2-D DFT

For 2D images the DFT is extended to a 2D DFT:

\[
F_{uv} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^{(-i2\pi \frac{u}{M})x} \cdot e^{(-i2\pi \frac{v}{N})y}
\]

The 2D DFT can be factorized into 2 x 1D DFT:

\[
f(x,y) \rightarrow F(x,v) \rightarrow F(u,v)
\]

rows \quad columns

Invariance of DFT:

If an object is shifted in the image domain then in the Fourier domain the amplitude of Fourier coefficients remains unchanged (invariance w.r.t. to image domain shift).

The Fourier-Mellin transform

Rotation and scaling performed in image domain result in a shift in the LogPolar domain of Fourier coefficients:

- Original \( F_k^{polar} = A_k \cdot e^{-i\theta_k} \), \( \ln F_k = \ln A_k + i \cdot \theta_k \)
- Scaled \( s \cdot F_k = s \cdot A_k \cdot e^{-i\theta_k} \)
- Scaled and rotated \( e^{-i\alpha} (s \cdot F_k) = s \cdot A_k \cdot e^{-i(\theta_k + \alpha)} \)
- LogPolar \( \ln(e^{-i\alpha} (s \cdot F_k)) = \ln s + \ln A_v - i(\theta_v + \alpha) \)
The Fourier-Mellin transform (2)

A second Fourier transform applied to the LogPolar transformed Fourier coefficients leads to the invariancy of new coefficient’s amplitudes w.r.t. scaling and rotation performed in the image domain.

\[
DFT[\ln F_k] = [B_k \cdot e^{-i\rho_k}], k = 0,\ldots, M - 1
\]

\[
DFT[\ln(e^{-i\alpha (s \cdot F_k))}] = [B_k \cdot e^{-i(\rho_k + \sigma)}], k = 0,\ldots, M - 1
\]
1. A sequence of patterns

Bayesian approach

Optimum pattern sequence search – to find the maximum value of a product of prior probability distributions:

\[ p(\Omega \mid C) = \frac{p(\Omega)p(C \mid \Omega)}{p(C)} = \alpha \cdot p(\Omega)p(C \mid \Omega) \]

with two stochastic processes:

\[ \Omega = (\Omega^1, \Omega^2, ..., \Omega^N) \] - sequence of classes

\[ C = (c^1, c^2, ..., c^N) \] - sequence of observed features

Let \( k \) be the number of classes and \( n \) - the length of the feature vector. Then in general the Bayes classifier would require: \( k^N \) prior probability densities, \( k^N \) conditional probability densities of \( n \times N \) - dimensional stochastic vectors.

Simplification \( \rightarrow \) a first-order Markov process.

First-order Markov process

Two plausible simplifications:

1. Individual observations are stochastically independent

\[ p(C \mid \Omega) = \prod_{i=1}^{N} p(c^i \mid \Omega^i = \Omega^k) \]

Instead of \( k^N \) number of \( n \times N \)-dimensional distributions, only \( k \) \( n \)-dimensional pdf-s are needed.

2. The classification of observation depends only on its direct predecessor (first-order Markov process), i.e.

\[ p(\Omega) = p(\Omega^1)p(\Omega^2 \mid \Omega^1)p(\Omega^3 \mid \Omega^2) \cdots p(\Omega^N \mid \Omega^{N-1}) \]

Instead of \( k^N \) prior probabilities it is sufficient to define \( k^2 \) transition probabilities \( P(\Omega_i \mid \Omega_j) \) and \( k \) probabilities \( P(\Omega_k) \).
Dynamic Programming

**Dynamic Programming (DP)** is an optimization approach for detecting a best sequence of decisions, assuming a first-order Markov process.

- It operates dually on a **cost measure** rather than on probability.
- It is a simplification of the Bayesian approach as only one **stochastic process** is considered (the decision sequence).

Basic assumption in DP: every decision regarding a sub-sequence is made in an optimal way (i.e. no backtracking).

This puts conditions onto the **cost function**:

- monotonic (cost is non decreasing w.r.t. decision steps),
- separable (into previous costs and new added costs).

---

**Model (problem graph)**

The **problem graph** in DP represents:

1. The transition costs (dual to \( p(\Omega_i|\Omega_{i-1}) \)),
2. A single observation is „accepted“ in given node of the problem graph, \( p(S_{k(i)}|\Omega^i) = 1 \), and \( p(S_j|\Omega^i) = 0 \), for \( j \neq k(i) \)

Example:

![Decision Graph](image)

<table>
<thead>
<tr>
<th>Node ( i )</th>
<th>Observation ( k(i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>T</td>
</tr>
<tr>
<td>n2</td>
<td>#</td>
</tr>
<tr>
<td>n3</td>
<td>A</td>
</tr>
<tr>
<td>n4</td>
<td>A</td>
</tr>
<tr>
<td>n5</td>
<td>E</td>
</tr>
<tr>
<td>n6</td>
<td>#</td>
</tr>
<tr>
<td>n7</td>
<td>B</td>
</tr>
</tbody>
</table>

**The decision space**

The solution of DP search - a path in the decision space with lowest cost. Decision space - a 2-D array indexed by the decision step and a node of the problem graph.
Decision costs

**A single decision:** how to extent a search path by a transition step from current to next node.

**Cost function**

The **prior cost function** (no observation, only transition costs):

\[ R(n, j) = \min_i \{ R(n-1, i) + r(i, j) \} \]

The **posterior cost function** (an observation sequence is given):

\[ R(n, j) = \min_i \{ R(n-1, i) + r(i, j) + r(S_n, j) \} \]

- \( r(i, j) \) - transition costs of arc \( i \rightarrow j \).
- \( r(S_n, j) \) – costs of accepting the \( n \)-th observation \( S_n \) in node \( j \) of the problem graph.

**Decision space**

**Example (cont.)**

In the **prior case** (no observation) DP search finds the path

\((0, n2)-(1, n7)-(2, n5)-(3, n4)-(4, n1)-(5, n3)-(6, n6)\)

which is of lowest possible cost (6), whereas other complete paths between \((0, n2)\) and \((n, n6)\) are of cost: 11, 12, 13 and 14.

In the **posterior case**, if observation is #BEATA# the solution is the same, while other paths are of infinite costs.

If observation is #B# the solution is:

\((0, n2)-(1, n7)-(2, n5)\).
DP-based object recognition

Example.
Using DP search provide a matching of the symbol chain (observed sequence) “adzete” with the model sequence “dzień”.

Solution. The nodes in the problem graph should represent possible decisions: accept given letter, replace a letter, insert a letter, delete a letter. All decisions (transitions to next node) except of the „delete a letter“-nodes, move to next letter in the observation sequence (and accept it or not).

The problem graph should allow a path of decisions: insert a, accept d, accept z, delete i, accept e, replace ń by t, insert e.

2. A structure of patterns

Design problems:
1. Make image segmentation – get data A;
2. Set the object model \( M_k \),
3. Design the matching method for segment to object parts,
4. Design the judgement rule for \( \theta \) these correspondences.
Optimum object recognition

Object detection — determine the image structure corresponding to an object given by model $\Omega_k$ (or $M_k$).

Object localisation in space — determine the location $(t, R)$ in space (3D) or plane (2D) — 6 parameters (3 translations and 3 rotations) (3-D), or 3 parameters (2 translations and 1 rotation).

Optimum recognition strategies

1. Systematic search (check all instances):
\[
\{M_k, R_k, t_k\} = \arg \max_{\{M_k, R_k; t_k\}} \theta[I(M_k(R_k; t_k)(A))];
\]

2. Bayes classifier: with known priors: $p(M_k(R_k; t_k))$, $p(A|M_k(R_k; t_k))$, the Bayes classification rule is:
\[
\{M_k, R_k, t_k\} = \arg \max_{\{M_k, R_k; t_k\}} p(M_k(R_k; t_k) | A).
\]

where $p(M_k(R_k; t_k) | A) \sim p(A | M_k(R_k; t_k)) p(M_k(R_k; t_k))$

Heuristic strategies

The systematic search and Bayes approach to object recognition are both difficult to implement, due to a huge complexity.

The hypothesize-and-test” strategy

**REPEAT**

(“Hypothesis generation”) Set initial hypotheses (object instances) by matching model parts with segments: $I^{(i)}(P_{i}^{(i)} \in M_k; O_{i}^{(i)} \in A)$. For every $I^{(i)}$ estimate its parameters: $a^{(i)} = (R_k; t_k)^{(i)}$

(“Hypothesis verification”) FOR every hypothesis $I^{(i)}$ DO Generate a full list of segment–to-parts matches $O^{(i)} \in A$.

IF the number of matches for parts of model $M_k$ is sufficiently large THEN the instance $I^{(i)}(M_k, O^{(i)})$ exists AND estimate its location parameters $a^{(i)} = (R_k; t_k)^{(i)}$ AND eliminate from $A$ the segments $O^{(i)}$. UNTIL more than $W\%$ of segments are still available in $A$. 

W. Kasprzak: ElASR 7. Model-based object recognition
A* graph search

Algorithm A* (informed graph search) allows an efficient search. Nodes are selected for expansion based on the cost function:

\[ f(n) = g(n) + h(n) \]

where \( g(n) \) – real costs of the path from start node to node \( n \),
\( h(n) \) – expected cost of remaining path from \( n \) to a terminal node.

Due to \( h(n) \) we call the search to be informed, as it allows a goal-driven selection of the next node.

The heuristics \( h(n) \) is admissible if it is an optimistic estimation of real remaining costs \( h^*(n) \) for every node \( n \), i.e.:
\[ h(n) \leq h^*(n) \]

In practice \( h(n) \) should be admissible and consistent: for each path from \( n \) to \( n' \) it should hold:
\[ h(n) \leq c(n, n') + h(n') \]

where \( c(n, n') \) is the real transition cost between node \( n \) and \( n' \).

A* graph search (2)

1 INIT: Get the start node \( s \) and put it in the set OPEN. Set \( f(s) = 0 \), \( g(s) = 0 \).
2 Get from OPEN next node \( n \) with smallest cost \( f(n) \) und put it in CLOSED.
3 IF \( n \) is a terminal node
   THEN stop and return \{ \( g(n) \) and path from \( s \) to \( n \) \}.
4 Get the successor nodes of node \( n \) – denote them: \( n_1', \ldots, n_k' \).
5 FOR each successor node \( n_1, \ldots, n_k \) DO:
   compute a cost: \( g_i = g(n) + c(n, n_i) \).
6 FOR each node \( n_1', \ldots, n_k' \):
   A IF \( n_i' \)-equivalent is not in OPEN nor CLOSED yet
      THEN add it to OPEN and set:
      \[ g(n_i') = g_i, \quad f(n_i') = g_i + h(n_i') \].
   B IF \( n_i' \)-equivalent is in OPEN or CLOSED already AND \( g(n_i) > g_i' \)
      THEN set \( g(n_i') = g_i, \quad f(n_i') = g_i + h(n_i') \),
      delete the path from \( s \) to \( n_i' \),
      cancel the \( n_i' \)-equivalent node and
      add \( n_i' \) to OPEN.
7 REPEAT from step 2.
3. Generic object model

An instance of an object class is distinguished by a parameter vector (also called attributes), e.g. with parameters for size, localisation, shape.

We call this a **generic object model**.

The object instance space is higher than in fixed-size object case. Such object recognition problem will be expressed here in terms of **incremental state estimation**:

- the unknown state parameters are estimated from a known observation sequence, while satisfying some criterion;
- originating from estimation theory: ML (maximum likelihood, MAP (maximum posterior), LSE (least square error), MMSE (minimum mean square error).

---

**Example**

Projection conditions for on-road 3-D vehicle objects

A simple generic vehicle model

Object state: \( s = s_{\text{Localisation}} \cup s_{\text{Shape}} \)

\[
s_{\text{Localisation}} = (t_x, t_z, \phi)^T, \quad s_{\text{Shape}} = (\text{Width}, \text{Height}_1, \text{Height}_2, \text{Length}_1, \text{Length}_2)^T
\]
Example (2)

The model-to-image projection $h(s)$:

Measurement $z$: image segmentation and finding the best match between projected model edges and detected line segments

$z = h(s) + w$

$z$ consists of line segments:

$z = \{z_1, \ldots, z_n\}$,  \hspace{1cm} $Z_i = \begin{bmatrix} mx_i \\ my_i \\ \phi_i \\ l_i \end{bmatrix}$

Iterative MAP estimation

To find an MAP estimate of the unknown state $s$ given observations it would require a very complex prior probability distribution

$p(z_1, z_2, \ldots, z_n \mid s)$

and an extensive search in a high-dimensional space.

We solve it approximately by a gradient descent (iterative) method:

• start with initial estimate of the state $\hat{s}_0$ and covariance matrix $P_0$.

• In every iteration a new estimate $\hat{s}_{0(t)}$ and covariance matrix $P_{0(t)}$ are generated that are better than all previous estimates.

The prior pdf-s: $p(s \mid \hat{s}_0) \sim N(\hat{s}_0, P_0)$,  \hspace{1cm} $p(z \mid s) \sim N(h(s), R)$

MAP estimate: to maximize the posterior:

$p(s \mid z, \hat{s}_0) = \frac{1}{C} \cdot p(z \mid s) \cdot p(s \mid \hat{s}_0) \sim \exp(-\Gamma(s))$

i.e. to minimize:

$\Gamma(s) = \frac{1}{2} [z - h(s)]^T \cdot R^{-1} \cdot [z - h(s)] + \frac{1}{2} (s - \hat{s}_0)^T \cdot P_0^{-1} \cdot (s - \hat{s}_0)$
Iterative MAP estimation (2)

The objective: \( \nabla \Gamma(s) = H(s) \cdot R^{-1} \cdot [h(s) - z] + P_0^{-1} \cdot (s - \hat{s}_0) = 0 \)

with \( H(s) = \frac{\partial h}{\partial s} \)

Iterative MAP

INIT: \( s_0(1), P_0(1) \).

REPEAT FOR \( t = 1, \ldots, N \)

1. Perform image segmentation – it gives the measurement \( z(t) \).
2. Estimate: \( s_1(t) = s_0(t) - \mu P_0(t) H(s(t)) \cdot R^{-1} \cdot [h(s_0(t)) - z(t)] \)
   \( P_1(t) = (1 - \mu) P_0(t) \)
   where \( \mu \in <0, 1> \) - is a small “forgetting” coefficient.
3. Set for next time point (simple prediction): \( s_0(t+1) = s_1(t), P_0(t+1) = P_1(t) \).

W. Kasprzak: EIASR 7. Model-based object recognition

EIASR
Speech signal pre-processing
Lecture 8
1. Digital audio

**Pulse code modulation (PCM)** - the common type of digital audio recording:
- Sampling rate is 44,100 samples taken every second,
- Amplitude gets converted into a 16-bit integer: 65,536 (64K).
- Two channels of (stereo) data.

A general multimedia file format is **RIFF (Resource Interchange File Format)**, where a file consists of data **chunks**, identified by a predecessor ASCII name (of 4 bytes). **WAV** is an instance of RIFF.

---

<table>
<thead>
<tr>
<th>Base</th>
<th>Bytes</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>4</td>
<td>The text &quot;RIFF&quot;</td>
</tr>
<tr>
<td>0004</td>
<td>4</td>
<td>The size of sound data = file size - 8</td>
</tr>
<tr>
<td>0008</td>
<td>4</td>
<td>The text &quot;WAVE&quot;</td>
</tr>
<tr>
<td>000C</td>
<td>4</td>
<td>The text &quot;fmt &quot;</td>
</tr>
<tr>
<td>0010</td>
<td>4</td>
<td>Size of the format data (16 bytes)</td>
</tr>
<tr>
<td>0014</td>
<td>2</td>
<td>wf.wFormatTag (WAVE_FORMAT_PCM = 1)</td>
</tr>
<tr>
<td>0016</td>
<td>2</td>
<td>wf.nChannels (mono, stereo)</td>
</tr>
<tr>
<td>0018</td>
<td>4</td>
<td>wf.nSamplesPerSec</td>
</tr>
<tr>
<td>001C</td>
<td>4</td>
<td>wf.nAvgBytesPerSec</td>
</tr>
<tr>
<td>0020</td>
<td>2</td>
<td>wf.nBlockAlign</td>
</tr>
<tr>
<td>0022</td>
<td>2</td>
<td>wf.wBitsPerSample</td>
</tr>
<tr>
<td>0024</td>
<td>4</td>
<td>The text &quot;data&quot;</td>
</tr>
<tr>
<td>0028</td>
<td>4</td>
<td>The data size in this file</td>
</tr>
<tr>
<td>002C</td>
<td></td>
<td>Data</td>
</tr>
</tbody>
</table>
Human auditory system

1. Bone chain
2. Auditory nerves
3. Inner ear (cochlea)
4. Eardrum
5. Outer ear
6. Eustachian tube

The **cochlea** (a spiral of tissue filled with **liquid** and thousands of tiny **hairs**) is acoustically coupled to the **eardrum** by a series of tiny **bones**. The hairs on the outside of the spiral are longer than the hairs on the inside. The **longer hairs** resonate with **lower frequency** sounds, and the **shorter hairs** with **higher frequencies**.

---

2. Fourier Transform

The **Fourier transform (FT) family**:

1. **Aperiodic-Continuous** signal → **Fourier Transform**
   A signal extends to both positive and negative infinity without repeating in a periodic pattern (e.g. Gaussian curve).

2. **Periodic-Continuous** → **Fourier Series**
   E.g. sine waves, square waves, and any waveform that repeats itself in a regular pattern from negative to positive infinity.

3. **Aperiodic-Discrete** → **Discrete-time Fourier Transform**
   Signals are defined at discrete points and do not repeat themselves in a periodic fashion.

4. **Periodic-Discrete** → **Discrete Fourier Transform**
   Discrete signals that repeat themselves in a periodic fashion.
Fourier Series

Fourier series: the Fourier Transform for periodic signals in continuous time. It transforms the signal \( x(t) \) into an equivalent summation of sine and cosine waves, with frequencies that are multiples of a base frequency, \( f_0 = 1/T \), where \( T \) is the period of \( x(t) \):

\[
x(t) = a_0 + \sum_{k=1}^{\infty} (a_k \cdot \cos(2\pi k f_0 t) + b_k \cdot \sin(2\pi k f_0 t))
\]

The Euler relation: to represent a pair of sine and cosine waves with the same frequency by a complex number:

\[
e^{jx} = \cos(x) + j\sin(x)
\]

\[
M e^{j\theta} = M(\cos(\theta) + j\sin(\theta)) \leftrightarrow a + jb
\]

The Euler formula

Let the signal \( x(t) \) be some \( 2\pi \)–periodic function. Then its composition expressed by complex numbers is:

\[
x(t) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} c_k e^{j2\pi f_0 kt}
\]

\[
c_k = \begin{cases} 
\pi(a_k - ib_k), & \text{if } k \geq 0 \\
\pi(a_{|k|} + ib_{|k|}), & \text{otherwise}
\end{cases}
\]

where the Fourier series coefficients are:

\[
a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} x(t) \cos(2\pi f_0 kt) dt \\
b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} x(t) \sin(2\pi f_0 kt) dt \\
c_k = \int_{-\pi}^{\pi} x(t) e^{-j2\pi f_0 kt} dt
\]
Discrete Fourier Transform (DFT)

- For discrete-time signals we need to consider a finite set of sine and cosine waves only, as digital recordings have a finite length.
- The number of DFT (output) frequencies is the same as the number of (input) signal samples in the time domain.
- We can pretend that the function \( x(t) \) is periodic, and that the period \( T \) is the same as the time of recording \( M \) samples. The base frequency, \( f_0 \), for the \( M \)-sample DFT, is:

\[
f_0 = \frac{f_s}{M} = \frac{1}{T}, \text{ where } f_s \text{ is the sampling rate.}
\]

**Definition.** Let \( x = [x_0, x_1, \ldots, x_{M-1}] \) be complex-valued samples of the discrete signal. The Discrete Fourier Transform of order \( M \) is:

\[
F_k = \sum_{t=0}^{M-1} x_t (e^{-i2\pi k/M})^t, \quad k = 0, \ldots, M-1
\]

DFT (1)

**DFT in matrix form:** \( F = D_M \cdot x \)

\[
\begin{bmatrix}
F_0 \\
F_1 \\
F_2 \\
\vdots \\
F_{M-1}
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & m & m^2 & \cdots & m^{M-1} \\
1 & m^2 & m^4 & \cdots & m^{2(M-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & m^{M-1} & m^{2(M-1)} & \cdots & m^{(M-1)^2}
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
\vdots \\
x_{M-1}
\end{bmatrix}
\]

\[m = e^{-i2\pi/M}\]

Complex- or real-valued input

In general the DFT converts a sampled complex-valued function of time into a sampled complex-valued function of frequency. Usually we operate on real-valued input functions, i.e. all the imaginary parts of the input signal are assumed to be zero.
Inverse DFT

The DFT matrix $D_M$ is a symmetric matrix. The inverse matrix exists, which is a complex conjugate of $D_M$:

$$D_M^{-1} = \frac{1}{M} D_M^*$$

$$D_M^* = \begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & m^{-1} & m^{-2} & \cdots & m^{-(M-1)} \\
1 & m^{-2} & m^{-4} & \cdots & m^{-2(M-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & m^{-(M-1)} & m^{-2(M-1)} & \cdots & m^{-(M-1)^2}
\end{pmatrix}$$

$$m = e^{-i\frac{2\pi}{M}}$$

Inverse DFT:

$$x = \frac{1}{M} D_M^* \cdot F$$

The Nyquist frequency

The Nyquist frequency

The frequency index $(M/2)$ is a special case: it corresponds to the Nyquist frequency, which is always half the sampling rate in any digital PCM recording.

For example, the Nyquist frequency in a typical CD audio recording is: $(44,100 \text{ Hz})/2 = 22,050 \text{ Hz}$.

The Nyquist frequency is the highest frequency that a PCM digital audio recording can reproduce.

Signals above the Nyquist frequency are filtered out before the signals are digitally sampled, in order to avoid aliasing problems.
Fast Fourier Transform (FFT)

**FFT** (*Cooley* and *Tukey* proposed the FFT algorithm in 1965)

The basic idea of the Fast Fourier Transform is that the DFT can be converted into a set of DFT-s of size 2.

FFT assumes that the number of discrete samples is a **power of two**, i.e. $M = 2^n$.

The DFT requires $O(M^2)$ operations, whereas FFT is of complexity $O(M \cdot \log_2(M))$.

Two equivalent FFT algorithms are distinguished:
1. FFT with decimation in the time domain,
2. FFT with decimation in the frequency domain.

**FFT with decimation in frequency**

Processing flow for $M=8$:
**FFT**

FOR \( j = 0, \ldots, k-1 \) DO (for every iteration)

FOR \( u = 0, \ldots, 2^j - 1 \) DO (for every group)

FOR every pair \((a, b)\) (\( v = 0, \ldots, 2^{k-j-1} - 1 \), of group \( u \)) DO

\[
X^{(j+1)}[a] = X^{(j)}[a] + m^u \cdot X^{(j)}[b];
\]

\[
X^{(j+1)}[b] = X^{(j)}[a] - m^u \cdot X^{(j)}[b];
\]

FOR \( s=0, \ldots, M-1 \) DO

Reverse the bit order of binary digits: \( F_s = X^{(j)}_z \)

with \( s = (s_{k-1} \ldots s_0)_2 \)

\( z = (s_0 \ldots s_{k-1})_2 \)

---

**3. Signal filtering**

A distorted measurement of a source signal is modeled as:

\[
f = s \otimes g + n
\]

where: \( f \) – measured signal, \( s \) – the ideal, non-distorted source, \( g \) – an unknown distortion convoluted with the source, \( n \) – an unknown additive noise.

First we concentrate on additive **noise reduction**, when

\[
f = s + n
\]

- High-frequency noise can be cancelled by applying a **low-pass filter** with appropriate cut-off frequency.
- Gaussian noise can be reduced by applying a **Gaussian smoothing filter**.
- Structured noise has to be identified first. For example by the method of **spectral subtraction**.
Noise reduction

Noise reduction by spectral subtraction
if the noise characteristics is unknown, but it can be measured.

Iteratively estimate the Fourier coefficients of the noise signal in frames which obviously include only the noise signal:

\[ |\hat{N}_{m,j}|^2 = (1 - \gamma) |\hat{N}_{m-1,j}|^2 + \gamma |F_{m,j}|^2 \]

where the parameter \( \gamma \approx 0.2 \) for frames without speech and \( \gamma = 0.0 \) for frames with speech.

Subtract the estimated noise energy from the total signal energy in frames, where we detect speech:

\[ |\hat{S}_{m,j}|^2 = \begin{cases} |F_{m,j}|^2 - \alpha |\hat{N}_{m,j}|^2 : \text{if} \; |F_{m,j}|^2 - \alpha |\hat{N}_{m,j}|^2 > \beta |F_{m,j}|^2 \\ \beta |F_{m,j}|^2 : \text{otherwise} \end{cases} \]

where the parameters \( \alpha \approx 0.9, \beta \approx 0.15 \).

Pre-emphasis filter

The goal of "pre-emphasis" is to strengthen the higher frequencies (is performed in the time domain):

\[ f_t^* = f_t - \varphi \cdot f_{t-1} \], where \( \varphi \in < 0.9, 1.0 > \).

The magnitude part of the frequency characteristics:

Example:
A spectrogram before and after pre-emphasis:
Auto-correlation of signal

For a window of size $N$ we compute normalized auto-correlation of signal samples with its shifted samples (by $k$).

Starting with $m$-th sample the $k$-shifted normalized auto-correlation is:

$$r_k^{(m)} = \frac{\sum_{n=m}^{m+N-k-1} f_n f_{n+k}}{|[f_n]||[f_{n+k}]|}$$

The maximum of normalized auto-correlation values always appears for $k=0$ and is equal to 1.

For a voiced speech part local maxima appear every $k_0$ samples: $r_k^{(m)}$, $r_{k+jk_0}$; e.g. for some $k_0$, $2k_0$, $3k_0$, etc., because many strong harmonic frequencies exist in this part.

W. Kasprzak: EIASR 8. Speech signal pre-processing

Base frequency

By normalized mutual correlation we shall mean a correlation factor between two consecutive signal frames.

Let the frames of signal samples are given (in vector form):

$$f(m-k, m-1) = [f_{m-k}, \ldots, f_{m-1}]$$
$$f(m, m+k-1) = [f_m, \ldots, f_{m+k-1}]$$

The normalized mutual correlation of these two vectors is:

$$\rho_m(k) = \frac{f(m-k, m-1) \cdot f(m, m+k-1)}{|f(m-k, m-1)| \cdot |f(m, m+k-1)|}$$

Start with $k=2$ and compute the correlation factor for different values of $k$. In a voiced part of the speech the maximum of normalized mutual correlation will be at $k$ that corresponds to the base period (and base frequency $F_0$) of the speaker.

W. Kasprzak: EIASR 8. Speech signal pre-processing
1. Speech features

Frame-based speech signal features

1. Mel-frequency cepstral coefficients (MFCC), extended by their first derivatives in time.

or

2. Speech features based on Linear Predictive Coding (LPC), e.g. LPCC – linear predictive cepstral coefficients
Cepstrum (1)

The **cepstrum** of a signal \( x[n] \) is the result of a homomorphic transformation:

\[
cepstrum(x) = F^{-1}(\log |F(x)|),
\]

where \( F \) is the discrete-time Fourier Transform (DFT) for MFCC or the \( Z \) transform for LPCC.

**Note:** spectrum \( \rightarrow \) spec | trum \( \rightarrow \) ceps | trum \( \rightarrow \) cepstrum

**MFCC:**

\[
\begin{align*}
\text{Signal frame} & \rightarrow X = \text{DFT}(x) & \rightarrow \text{mfc} = \log |X| & \rightarrow \text{c} = \text{DFT}^{-1}(\text{mfc}) & \rightarrow \text{cepstrum}
\end{align*}
\]

**LPCC:**

\[
\begin{align*}
\text{Signal frame} & \rightarrow A = \text{Z}(\text{LPC}(x)) & \rightarrow h = \log(1/A) & \rightarrow c = Z^{-1}(h) & \rightarrow \text{cepstrum}
\end{align*}
\]

Cepstrum (2)

Why are **cepstrum features** useful for speech recognition?

- The cepstrum features characterizing the (impulse response of the) **vocal tract** are located near the “zero” feature \( k=0 \); whereas the input impulse components, corresponding to the **larynx-modulated oscillations** (that are not useful for speech recognition) are located at higher values of \( k \) (“longer” cepstrum time), where the cepstrum features achieve a maximum value;

- The useful features can be separated from the others by selecting an appropriate number of them, starting from \( k=0 \), or by additional multiplication, called **liftering**.

- The speech impulse response can also be separated from the acquisition channel’s (microphone) response by using **centered cepstrum** features.
2. MFCC

1. Short-time Fourier Transform (STFT)

A windowed DFT for every frame $\tau$ of the input signal:

$$F(k, \tau) = \sum_{t=0}^{M-1} (x[\tau + t] \cdot e^{-i2\pi kt/M} \cdot w_\tau[t]) \quad , k=0, 1, ..., M-1$$

Window functions $w[t]$

1. Rectangular window
2. Triangle window
3. Hamming window

$$w_\tau[t] = \begin{cases} 0.54 - 0.46 \cos\left(\frac{2\pi t}{M}\right), & \text{for } t = \{0, 1, ..., M-1\} \\ 0 , & \text{otherwise} \end{cases}$$

Example. $x[n]$ is the sum of two sinus functions uniformly sampled from 0 to $2\pi$ by 128 samples:

$$x[n] = \sin(2\pi n/5) + \sin(2\pi n/12) \quad , n=0,1,2,\ldots,127.$$

Single frame (rectangular window applied)

(Hamming window applied)
Windowing example (2)

Example (cont.) Magnitude of Fourier coefficients:
With rectangular window. With Hamming window

Window functions

Conclusion:

\[ f \times w \]

\[ \text{Mag}[\text{DFT}(w)] \] \hspace{1cm} \[ \text{Mag} [\text{STFT}(f \times w)] \]
Spectrogram

Power of Fourier coefficients (squared magnitude)

\[ FC(k, \tau) = \left| F(k, \tau) \right|^2 = \left| \sum_{t=0}^{M-1} (x[\tau + t] e^{-i2\pi kt/M} \cdot w(t)) \right|^2, \quad k = 0, \ldots, M-1 \]

Mel frequency scale

Non-linear response of the human ear to the frequency components in the audio spectrum: differences in frequencies at the low end (< 1 kHz) are easier detectable than differences of the same magnitude in the high end of the audible spectrum.

Approach: a non-linear frequency analysis performed by the human ear - the higher the frequency the lower its resolution

MEL scale (empirical result):

\[ f_{mel} = 2595 \log \left( 1 + \frac{f}{700[Hz]} \right) \]
MFC

Mel frequency coefficients (MFC)

Triangular filters are located uniformly in the Mel frequency scale:

\[
MFC(l, \tau) = \sum_{k=0}^{M-1} [D(l, k) \cdot FC(k, \tau)] \quad l = 1, \ldots, L
\]

The MFC value associated with each bin corresponds to a weighted average of the power spectral values in the particular frequency range specified by the shape of the filter.

MFCC

The Mel-frequency cepstrum coefficients are computed by the homomorphic transformation

\[
MFCC(h) = FT^{-1}\{\log MFC\{FT\{h\}\}\}, \quad \text{for} \quad h = x \otimes w
\]

The last step is the inverse Fourier Transform of logarithmic Mel frequency coefficients:

\[
MFCC(k, \tau) = \sum_{l=0}^{L-1}[\log MFC(l, \tau) \cdot \cos\left(\frac{k \cdot (2l + 1) \pi}{2L}\right)] \quad k = 1, \ldots, K
\]

Centered MFCC

\[
MFCC_{centered}(k, \tau) = MFCC(k, \tau) - \text{mean}\{MFCC(k, \tau)\} \quad \tau = 1, 2, \ldots
\]

\[
k = 1, \ldots, K
\]
Delta features

**Energy feature**
Additional feature - the total energy of signal in a single frame:
\[ E(\tau) = \log(\sum_{i=1}^{M} x_i^2) \]

**Gradients of features in time (”delta” features)**
A schematic view of spectrograms for different phoneme types: single vowels (left), diphthongs (middle), plosives (right).

A linear regression in 5 consecutive frames is applied to find delta coefficients „d” (of MFCCs and energy feature „c”):
\[ d(\tau) = \frac{2c(\tau + 2) + c(\tau + 1) - c(\tau - 1) - 2c(\tau - 2)}{10} \]

---

Feature set

<table>
<thead>
<tr>
<th>Energy</th>
<th>MFCC</th>
<th>Delta energy</th>
<th>Delta MFCC</th>
<th>General features per frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>c0</td>
<td>E_mel</td>
<td>c19</td>
<td>c20</td>
<td>c38</td>
</tr>
<tr>
<td>c1</td>
<td>mfcc_1</td>
<td>c20</td>
<td>c21</td>
<td>c39</td>
</tr>
<tr>
<td>c2</td>
<td>mfcc_2</td>
<td>c21</td>
<td>c22</td>
<td>c40</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>c22</td>
<td>c23</td>
<td>c41</td>
</tr>
<tr>
<td>c18</td>
<td>mfcc_18</td>
<td>c23</td>
<td>c24</td>
<td>c42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c24</td>
<td>c25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c25</td>
<td>c26</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c26</td>
<td>c27</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c27</td>
<td>c28</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c28</td>
<td>c29</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c29</td>
<td>c30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c30</td>
<td>c31</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c31</td>
<td>c32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c32</td>
<td>c33</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c33</td>
<td>c34</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c34</td>
<td>c35</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c35</td>
<td>c36</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c36</td>
<td>c37</td>
<td>c38</td>
</tr>
</tbody>
</table>

**General features per frame**
(Total energy, mean and variance, norm. max. auto- correlation, low-band ratio)

W. Kasprzak: EIASR 9. Acoustic feature detection
3. LPC and LPCC

The Z transform is a discrete-time signal transform, which is dual to the Laplace transform of continuous-time signals, that means a probing of signal by sinusoids and (decaying) exponentials:

\[ X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n} \]

and \( z \) is a complex number: \( z = re^{j\omega}, \quad r = e^{-\sigma} \).

The synthesis model of human speech (in \( z \)-domain) consists of:

- an excitation source \( E(z) \) on the input,
- a linear filter with transmittance \( H(z) \),
- the speech signal \( X(z) \) on its output;

(the signals and the filter are represented by their transforms in the complex-valued domain \( z \)).

Speech synthesis model

Let us denote by \( H(z) \) the transmittance of the filter (the \( z \) transform of its frequency response \( h[n] \)). There are obvious relations in the \( z \) domain:

\[ X(z) = H(z)E(z), \quad E(z) = A(z)X(z) \]
IIR filter

A **digital IIR filter** is characterized by a recursive equation:

\[ x[n] = b_0 e[n] + b_1 e[n-1] + b_2 e[n-2] + \cdots + b_p e[n-p] + a_1 x[n-1] + a_2 x[n-2] + \cdots + a_m x[n-m] \]

The \( n \)-th output sample, \( x[n] \), is computed from the current and previous input samples and previous output samples. In short:

\[ x[n] - \sum_{k=1}^{m} a_k x[n-k] = \sum_{k=0}^{p} b_k e[n-k] \]

A corresponding description in the z-domain is:

\[
X(z) = E(z) \frac{\sum_{k=0}^{p} b_k z^{-k}}{1 - \sum_{k=1}^{m} a_k z^{-k}} \quad H(z) = \frac{\sum_{k=0}^{p} b_k z^{-k}}{1 - \sum_{k=1}^{m} a_k z^{-k}}
\]

---

LPC

The **Auto-Regressive** (AR) model assumes that the numerator is 1:

Thus in the AR model the \( n \)-th output sample, \( x_n \), is estimated only on \( m \) previous output samples and current input sample as:

\[ x[n] = e[n] + a_1 x[n-1] + a_2 x[n-2] + \cdots + a_m x[n-m] \]

In short:

\[ x_n = e_n + \sum_{k=1}^{m} a_k x_{n-k} \]

Ideally, for voiced parts the vocal tract is cyclically fed by a Dirac delta impulse. Then: \( e_0=1, \ e_n=0 \), for short-time frames.

Thus, the \( n \)-th speech sample (in a frame) is estimated as a linear combination of the previous \( m \) samples:

\[ \hat{x}_n = \sum_{k=1}^{m} a_k x_{n-k} \]
**Auto-correlation method for LPC**

The task is to compute the parameters, \( \{ a_k | k=1, \ldots, m \} \), for every signal frame. By the LSE approach, for given frame, we have:

\[
\varepsilon = \sum_{n=0}^{n_1} (x_n - \hat{x}_n)^2 \quad \frac{\partial \varepsilon}{\partial a_i} = \sum_{n} \left(x_n - \sum_{k} a_k x_{n-k}\right) 2x_{n-i} = 0
\]

where \( n_0, n_1 \) are training sample indices in given frame.

We get \( m \) equations with \( m \) unknowns:

\[
\sum_{k} a_k \sum_{n} x_{n-k} x_{n-i} = \sum_{n} x_n x_{n-i} \quad i = 1,\ldots,m
\]

By introducing the first \( m+1 \) auto-correlation coefficients:

\[
r_{i-|k|} = \sum_{n=0}^{M-1} x_n x_{n+i-k} = \sum_{n} x_{n-k} x_{n-i}
\]

the equation system takes the form:

\[
\sum_{k=0}^{m} a_k r_{i-|k|} = r_i \quad i = 1,\ldots,m
\]

W. Kasprzak: EIASR 9. Acoustic feature detection

---

**LPC computation**

\[
\begin{pmatrix}
  r_0 & r_1 & r_2 & \cdots & r_{m-1} \\
  r_1 & r_0 & r_1 & \cdots & r_{m-2} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  r_{m-1} & r_{m-2} & r_{m-3} & \cdots & r_0 \\
\end{pmatrix}
\begin{pmatrix}
  a_1 \\
  a_2 \\
  \vdots \\
  a_m \\
\end{pmatrix}
= 
\begin{pmatrix}
  r_1 \\
  r_2 \\
  \vdots \\
  r_m \\
\end{pmatrix}
\]

\[
Ra = r
\]

The matrix \( R \) is a Toeplitz matrix (it is symmetric with equal diagonal elements). Due this Toeplitz property an efficient algorithm is available for computing \( a \) without computing the inverse matrix \( R^{-1} \).

**Alternative method**

The *Levinson-Durbin algorithm* is an iterative method for the computation of LPC parameters.
The Levinson-Durbin algorithm

$E$ represents the prediction error, $K_i$ – the reflection coefficients between consecutive parts of the acoustic tube, $a_j$ – the final prediction coefficients

INIT: $E_0 = r_0$, $i = 1$

FOR $i = 1$ to $m$

$$K_i = \frac{1}{E_{i-1}} \left( r_i - \sum_{j=1}^{i-1} \alpha_{j,i-1} r_{i-j} \right)$$

FOR $j = 1$ to $i - 1$

$$\alpha_{j,i} = \alpha_{j,i-1} - K_i \alpha_{i-j,i-1}$$

$$\alpha_{i,i} = K_i$$

$$E_i = (1 - K_i^2)E_{i-1}$$

FOR $j = 1$ to $m$

$$a_j = \alpha_{j,M}$$

LPC-based features

Basic parameters

The prediction parameters can itself be considered as a feature vector (usually the error value is also considered):

$$c_i = a_i \quad \text{for} \quad i = 0, \ldots, m; \text{where} \quad a_0 = 1; \text{and} \quad c_{m+1} = \varepsilon.$$

The number of prediction parameters is: $m \in \langle 12, 20 \rangle$.
Approximately for the sampling frequency $f_s$ [kHz]: $m \approx f_s + [2, 4]$.

Spectral features (smoothed signal spectrum)

By transforming the parameter vector into frequency domain we get a smooth spectrum of the signal frame. The required resolution in frequency domain is achieved by padding the parameter vector with zeros to get a vector with $M$ elements:

$$A_M = DFT([1, a_1, a_2, \ldots, a_m, 0, 0, \ldots, 0])$$
LPCC (1)

LPCC - the cepstral LPC

Recall, the speech synthesis filter function is transformed to the z-domain transmittance function:

\[ H(z) = \frac{1}{1 - \sum_{k=1}^{m} a_k z^{-k}} \]

The polynomial in the denominator part can be reorganized giving an all-pole transmittance function:

\[ H(z) = \frac{1}{1 - \sum_{k=1}^{m} a_k z^{-k}} = \frac{1}{\prod_{k=1}^{m} (1 - p_k z^{-1})} \]

Next we use the \( \ln \) - function and apply the inverse Z transform:

\[ c[1 : m] = Z^{-1}(\ln[H(z)]) = Z^{-1}\left(\sum_{k=1}^{m} \ln[p_k z^{-1}]\right) \]

LPCC (2)

A direct iterative method for computing the LPCC features

Instead of performing the particular steps of the cepstrum transformation of LPC coefficients, there exists an iterative method for a direct computation of LPCC features from the LPC coefficients.

For \( 1 \leq n \leq m \) (where \( m \) is the order of LPC transform):

\[ c[n] = -a_n - \sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right)c[n-k]a_k ; \quad n = 1, 2, ..., m \]

For \( n > m \):

\[ c[n] = -\sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right)c[n-k]a_k ; \quad n > m \]
1. Phonetic categories

**Phone (articulated sound)** - the smallest element in speech.

“A phone is a speech sound considered as a physical event without regard to its place in the sound system of a language” (Webster dictionary).

A phone may have different realisations, determined by: **tonality, duration, and intonation**. The **IPA (Int. Phonetic Association)** has defined some 100 phones, while at least 40 are required for a rough transcription.

**Phoneme** – a category of similar phones.

Even though no two speech sounds are identical, all of the phones classified into one phoneme category are similar enough to have the same meaning.
Groups of phonemes

Primary groups: **vowels, consonants** - a basic contrast – can a phoneme serve as a **syllable nucleus** or not?

The **mode of phonation:**

1. **Sonorants** – „buzzes“ - characterized mainly by **voicing** - the repetitive opening and closing of the vocal cords.
2. **Fricatives** – „hisses“ - generally non-voicing sounds.
3. **Plosives** – „pops“ - explosive sounds (including affricates).
4. **Silence** - phrase marker, breathing, mini-silences and closures before a plosive.

Vowels

1. **Monophthongs** (11 phonemes) - a single vowel quality.
   - **stressed** vowels: /A/, /E/, /@/, /i:/, /o/, /U/ (e.g. in the words *father, bet, bat, beet, above, boot*);
   - **reduced** vowels: /^/, /&/, /l/, /u/; (e.g. in the words *bus, above, bit, book*),
   - **r-coloured** vowels *E*: long /3r/, short /&r/ (as in *bird* and *butter*).
2. **Diphthongs** (6) - a clear change from start to end:
   /ei/ (e.g. *late*), /al/ (e.g. *bye*), />i/ (e.g. *boy*), /iU/ (e.g. *few*), /aU/ (e.g. *loud*), /oU/ *boat*. 
Consonants (1)

3. **Approximants** (4) – Semivowels – similar to vowels but more obstacles in the vocal tract than for the vowels:
   - **liquids**: the /l/ (e.g. "like") and the retroflex /ɾ/ (e.g. "red");
   - **glides**: the /j/ (e.g. "yes") and the /w/ (e.g. "won").

4. **Nasals** (3) - The airflow is blocked completely in the oral tract, but a lowering of the velum allows a weak flow through the nose:
   - /m/ as in "me", /n/ as in "new", /ŋ/ as in "sing".

5. **Fricatives** (9) - Weak or strong friction noises, when the articulators are close together to cause turbulence in the airflow:
   - **voiceless** /f/, /t/, /s/, /h/ as in: “fine”, “thing”, “sign”, “assure”, “hope”;
   - **voiced** /v/, /ð/, /z/, /ʒ/ as in: “voice”, “than”, “resign”, “vision”.

W. Kasprzak: EIASR 10. Phonetic speech model

Consonants (2)

6. **Plosives** (6) - Bursts or explosive sounds produced by complete closure of the vocal tract followed by a rapid release of the closure:
   - **unvoiced** /pʰ/, /tʰ/, /kʰ/ (as in “can”), and
   - **voiced**: /b/, /d/, /g/.

7. **Affricates** (2) - Plosives released with frication:
   - the /tʃ/ sound (like in "church"), the /dʒ/ (like in "judge").

The **IPA (International Phonetic Alphabet)** (c/o Department of Linguistics, University of Victoria, Victoria, British Columbia, Canada) is recognised as the international standard for the transcription of phonemes in all languages.

W. Kasprzak: EIASR 10. Phonetic speech model
**Articulation of vowels**

Articulation of vowels and /j/ according to: level of mouth opening and tongue location.

<table>
<thead>
<tr>
<th></th>
<th>Front</th>
<th>Central</th>
<th>Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>High (close)</td>
<td>normal „i”, „y” /iː/ , /j/</td>
<td></td>
<td>normal „u” /u/</td>
</tr>
<tr>
<td>High-mid (close-mid)</td>
<td>short „i” /ɪ/</td>
<td></td>
<td>short „u” /u/</td>
</tr>
<tr>
<td>Mid</td>
<td>normal „e” /ɛ/</td>
<td>short „e”, coloured „e” /æ/ , /ɜː/ , /ʌr/</td>
<td>„o” /o/</td>
</tr>
<tr>
<td>Low-mid (Open-mid)</td>
<td>a/e /æ/</td>
<td></td>
<td>short „a” /æ/</td>
</tr>
<tr>
<td>Low (open)</td>
<td></td>
<td>normal „a” /ə/</td>
<td></td>
</tr>
</tbody>
</table>

W. Kasprzak: EIASR 10. Phonetic speech model

---

**Articulation of consonants**

Classification of consonants according to phonetic categories and articulation area:

<table>
<thead>
<tr>
<th></th>
<th>Bilabial</th>
<th>Labio-dental</th>
<th>Dental</th>
<th>Alveolar</th>
<th>Post-alveolar</th>
<th>Retroflex</th>
<th>Palatal</th>
<th>Velar</th>
<th>Glottal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plosive</td>
<td>p /pʰ/</td>
<td>b /b/</td>
<td>t /t/</td>
<td>d /d/</td>
<td></td>
<td></td>
<td>k /kʰ/</td>
<td>g /ɡ/</td>
<td>closures /pʰ/ /kʰ/</td>
</tr>
<tr>
<td>Nasal</td>
<td>m /m/</td>
<td></td>
<td>n /n/</td>
<td></td>
<td></td>
<td></td>
<td>ng /ŋ/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Liquid approx.</td>
<td></td>
<td></td>
<td>l /l/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glide approx.</td>
<td></td>
<td></td>
<td>pol. „l” /w/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fricative</td>
<td>f /f/</td>
<td>w /v/</td>
<td>th /θ/</td>
<td>dz /dʒ/</td>
<td>s /s/</td>
<td>z /z/</td>
<td>sz /s/</td>
<td>ż /ż/</td>
<td>h /h/</td>
</tr>
<tr>
<td>Affricate</td>
<td></td>
<td></td>
<td>cz /tʃ/</td>
<td>dž /dʒ/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

W. Kasprzak: EIASR 10. Phonetic speech model
2. Spectral cues

**Formants** (energy concentrations in frequency bands) F0-F5

**Basic frequency (F0)** – the pitch

Usually: 80-200 Hz (man), 150-350 Hz (women)

**F1 can vary from 300 Hz to 1000 Hz**

The lower it is, the closer the tongue is to the roof of the mouth.

/i:/ has the lowest F1: \(~300 \text{ Hz}\); /A/ has the highest F1 \(~950 \text{ Hz}\).

**F2 can vary from 850 Hz to 2500 Hz**

The F2 value is proportional to the front or back position of the tongue tip. In addition, lip rounding causes a lower F2 than the case with unrounded lips.

/i:/ has an F2 of 2200 Hz, the highest F2 of any vowel; /u/ has an F2 of 850 Hz - the tongue tip is far back, and the lips are rounded.

---

**Formants**

**Distribution of main formants in vowels:**

<table>
<thead>
<tr>
<th></th>
<th>Front</th>
<th>Central</th>
<th>Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>i, /i/</td>
<td></td>
<td>u, /u/</td>
</tr>
<tr>
<td></td>
<td>F1 (\approx) 280 Hz</td>
<td>F1 (\approx) 310 Hz</td>
<td>F2 (\approx) 870 Hz</td>
</tr>
<tr>
<td></td>
<td>F2 (\approx) 2250 Hz</td>
<td>F2 (\approx) 450 Hz</td>
<td>F2 (\approx) 1030 Hz</td>
</tr>
<tr>
<td></td>
<td>F3 (\approx) 2900 Hz</td>
<td>F3 (\approx) 2250 Hz</td>
<td>F3 (\approx) 2380 Hz</td>
</tr>
<tr>
<td>High-mid</td>
<td>i (short) – y, /i/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1 (\approx) 400 Hz</td>
<td>F1 (\approx) 500 Hz</td>
<td>F1 (\approx) 500 Hz</td>
</tr>
<tr>
<td></td>
<td>F2 (\approx) 1900 Hz</td>
<td>F2 (\approx) 1500 Hz</td>
<td>F2 (\approx) 1050 Hz</td>
</tr>
<tr>
<td></td>
<td>F3 (\approx) 2550 Hz</td>
<td>F3 (\approx) 2500 Hz</td>
<td></td>
</tr>
<tr>
<td>Mid</td>
<td>e, /E/</td>
<td>c (short), /æ/</td>
<td>o, /o/</td>
</tr>
<tr>
<td></td>
<td>F1 (\approx) 560 Hz</td>
<td>F1 (\approx) 500 Hz</td>
<td>F1 (\approx) 640 Hz</td>
</tr>
<tr>
<td></td>
<td>F2 (\approx) 1770 Hz</td>
<td>F2 (\approx) 1500 Hz</td>
<td>F2 (\approx) 1190 Hz</td>
</tr>
<tr>
<td></td>
<td>F3 (\approx) 2490 Hz</td>
<td>F3 (\approx) 2500 Hz</td>
<td>F3 (\approx) 2390 Hz</td>
</tr>
<tr>
<td>Low-mid</td>
<td>a, /æ/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1 (\approx) 690 Hz</td>
<td>F1 (\approx) 640 Hz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F2 (\approx) 1660 Hz</td>
<td>F2 (\approx) 1190 Hz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F3 (\approx) 2490 Hz</td>
<td>F3 (\approx) 2390 Hz</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>open</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Typical spectrograms (1)

1. **Monophthong vowels**
   → a strong voicing, with stable formants

2. **Diphthong vowels**
   → A strong voicing but moving formants

3. **Approximants**
   Voiced but have less visible formants

Typical spectrograms (2)

4. **Nasals**
   → low energy and characteristic "zero" region :

5. **Fricatives**
   → have a high-frequency Gaussian region :

6. **Plosives**
   → An explosive burst of acoustic energy, following a short period of silence :

7. **Affricates**
   → A plosive followed by a fricative:
3. Sub-phonemes

**Pronunciation differences** (co-articulation effects): phones have a great influence on neighbour phones.

**Tri-phone model**: to split each phone into one, two, or three parts, depending on the typical duration of that phone as well as how much that phone will be influenced by surrounding phones.

**Example**. Tri-phones that represent the phoneme /E/ in "yes":
1. $\text{front}<E$ (/E/ in the context of a preceding front vowel),
2. $<E$ (/E/ in the middle with no contextual effects),
3. $E>$fric (/E/ in the context of a following fricative).

In general /E/ can be decomposed into 17 tri-phones:
• 8 for possible left context,
• 8 for possible right context, and
• 1 context-independent (centre phone).

---

**Context categories (1)**

1. **Front** - front vowels and similar approximants.
2. **Mid** - central vowels and similar approximants.
3. **Back** - back vowels and similar approximants.
4. **Sil** - silence.
5. **Nasal** - nasals.
6. **Retro** - retroflex approximant and retroflex (r-coloured) vowels.
7. **Fric** - fricatives and affricates.
8. **Other** - plosives and remaining approximants.
### Context categories (2)

<table>
<thead>
<tr>
<th>Possible previous phoneme type</th>
<th>Current phoneme</th>
<th>Possible next phoneme type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Front</td>
<td>$Mid</td>
<td>$Front</td>
</tr>
<tr>
<td>$Mid</td>
<td>$Back</td>
<td>$Mid</td>
</tr>
<tr>
<td>$Back</td>
<td>$Sil</td>
<td>$Back</td>
</tr>
<tr>
<td>$Sil</td>
<td>$Nasal</td>
<td>$Sil</td>
</tr>
<tr>
<td>$Nasal</td>
<td>$Retro</td>
<td>$Nasal</td>
</tr>
<tr>
<td>$Retro</td>
<td>$Fric</td>
<td>$Retro</td>
</tr>
<tr>
<td>$Fric</td>
<td>$Other</td>
<td>$Fric</td>
</tr>
<tr>
<td>$Other</td>
<td></td>
<td>$Other</td>
</tr>
</tbody>
</table>

8 context groups | Left | center | right |
3-part phoneme

left context $E$, $E >$, $E >$ right context
1. Speech variability

1. **Acoustic-phonetic variability** covers different accents, pronunciations, pitches, volumes, and so on.

2. **Temporal variability** covers different speaking rates.

3. **Syntactic variability** (of sentences)

A useful simplification is to treat them independently.

The **temporal variability** is easier to handle - an efficient algorithm is known as **Dynamic Time Warping**.

**Acoustic-phonetic variability** is more difficult to model, it covers acoustic feature schemes and phonetic modeling.

**Syntactic variability** deals with different word sequences representing the same sentence (meaning).

---

**Dynamic Time Warping**

Let $Y$ be a prototype (reference pattern), $X$ – current observation (input pattern). Both $Y$ and $X$ are sequences of simple patterns.

The distance $D(X; Y)$ between the input and a reference pattern is defined despite different lengths of sequences.

The goal of DTW is to find a path in the space of possible matches of both sequences, that is optimal w.r.t. the cost function, i.e.

$$ l = \arg \min \sum_{i} D(X, Y) $$

$D(X; Y)$ is defined as the sum of local distances, $d_{ij} = d(x_i, y_j)$, between corresponding elements in the solution path. The cost of a path leading to „state” $(x_i; y_j)$ is:

$$ R(x_i, y_j) = \min \left( R(x_{i-1}, y_{j-1}) + d(x_i, y_{j-1}) \right) $$
DTW - Example

Matching a 3-segment input pattern \((x_1 \ x_2 \ x_3)\) with a 4-element model pattern \((y_1 \ y_2 \ y_3 \ y_4)\).

An existing path can only be extended by one of the three allowed transitions:

- \(\text{state}(x_{i-1}, y_{i-1}) \rightarrow \text{state}(x_i, y_j)\)
- \(\text{state}(x_{i-1}, y_i) \rightarrow \text{state}(x_i, y_j)\)
- \(\text{state}(x_i, y_{i-1}) \rightarrow \text{state}(x_i, y_j)\)

Word recognition

In word recognition the question is: what is the most likely word represented by given acoustic sequence? A word consists of a sequence of phonemes.

\[
W = \arg \max_{\text{word}} P(\text{word} | \text{signal})
\]

From the Bayes rule we get:

\[
P(\text{word}|\text{signal}) = \alpha \cdot P(\text{signal}|\text{word}) \cdot P(\text{word})
\]

Two prior stochastic models are required:

1. the acoustic-phonetic model – \(P(\text{signal}|\text{word})\), and
2. the language model – \(P(\text{word})\).

In particular: the signal is represented by \(n\) feature vectors:

\[
P(\text{word} | c_1 c_2 \ldots c_n) = \alpha \cdot P(c_1 c_2 \ldots c_n | \text{word}) \cdot P(\text{word})
\]
Sentence recognition

In sentence recognition the question is: **what is the most likely word sentence represented by given sequence of words?** A sentence consist of words given in our lexicon.

\[ S = \text{arg}_{\text{sequence}} \max_P P(\text{sentence} | \text{words}) \]

*Sentence* is a meaningful sequence of words, *signal* is a sequence of observed words.

From the Bayes rule we get:

\[ P(\text{sentence} | \text{words}) = \alpha \cdot P(\text{words} | \text{sentence}) \cdot P(\text{sentence}) \]

Two prior stochastic models are needed:
1. the application model – \( P(\text{words} | \text{sentence}) \) and
2. the language model – \( P(\text{sentence}) \).

---

Hidden Markov Model

A **HMM** is defined as a 5-tuple: \( HMM = (S, Y, \Pi, A, B) \), where

- \( S = \{S_1, S_2, \ldots, S_N\} \) - set of states,
- \( Y \) - set of output symbols,
- \( \Pi \) - the start state probability vector,
- \( A = [a_{ij}] \) - the state transition probability matrix,
- \( B = [b_{ij}] \) - the output probability matrix.
HMM in word recognition

Output probability model

1. Discrete scalar - a single output symbol is observed in one state: \( P_j(O^t = Y_k \mid S_j) = b_{jk} \); \( j = 1,...,N; k = 1,...,M; t = 1,...,T \)

2. Discrete vector – a vector of symbols per state is observed:

\[
P_j(O^t = c^t \mid S_j) = \sum_{k=1}^{M} b_{jk} \cdot P_k(c^t)\]

3. Semi-continuous distribution – a mixture of Gaussian distributions – per class distributions

\[
P_j(O^t = c^t \mid S_j) = \sum_{k=1}^{M} b_{jk} \cdot p_k(c \mid \Omega_k) = \sum_{k=1}^{M} b_{jk} \cdot N(c; \mu_k, \Sigma_k)\]

4. Continuous (mixture of different distributions per state and class):

\[
P_j(O^t = c^t \mid S_j) = \sum_{k=1}^{M} b_{jk} \cdot p_k(c \mid \Omega_{jk}) = \sum_{k=1}^{M} b_{jk} \cdot N(c; \mu_{jk}, \Sigma_{jk})\]

HMM in word recognition (2)

Structure:
a left-to-right HMM

Actions:
- INSert,
- SUBstitute,
- DELete
Viterbi search (1)

Viterbi search - the goal is to find the best path of states in the HMM, w.r.t. the observed sequence:

$$(S^1 \ldots S^T) = \arg \max_{S^1 \ldots S^T} P(S^1 \ldots S^T | c^1 c^2 \cdot \cdot \cdot c^T)$$

This is done iteratively, from $t=1$ to $t=T$, while extending every partial path $(S^1 S^2 \ldots S^{t-1})$ in the HMM model, in a best way with respect to the observed sequence of feature vectors $(c^1 c^2 \ldots c^t)$, according to the dynamic programming principle:

$$P(S^1 \ldots S^{t-1} S^t | c^1 c^2 \ldots c^t) = P(c^t | S^t) \cdot \max_{S^t} (P(S^t | S^{t-1}) \cdot P(S^1 \ldots S^{t-1} | c^1 \ldots c^{t-1}))$$

In practice a logarithmic scale can be used:

$$\ln P(S^1 \ldots S^{t-1} S^t | c^1 c^2 \ldots c^t) = \ln P(c^t | S^t) + \max_{S^t} (\ln P(S^t | S^{t-1}) + \ln P(S^1 \ldots S^{t-1} | c^1 \ldots c^{t-1}))$$

W. Kasprzak: ElASR 11. Word and sentence recognition

Viterbi search (2)

The search space in Viterbi search – consistent with the dynamic programmimng search - is a 2-D space indexed by time (observed sequence) and HMM state indices.

Recall that in a discrete HMM:

$$a_{ij} = P_i(S_j | S_i); \quad b_{jk} = P(Y_k | S_j)$$
Viterbi search (3)

\[
G_0(\#) = \ln 1 = 0; \quad \psi_0(\#) = \{\} ; \quad \text{// Every path will start from \#}
\]

// Initialisation for all classes (non \# - HMM-states)

FOR \( j = 1, 2, \ldots, N \) DO:
\[
\psi_0(j) = \{\}; \quad G_0(j) = \ln \pi(S_j) + -\infty \quad \text{// No path can start here}
\]

FOR \( t = 1 \) TO \( T \) DO // Observation sequence

FOR \( j = 1 \) TO \( M \) DO // All output symbols

// Find the best edge in graph
\[
1) \quad G_t(j) = \ln P(c_t | S_j) + \max_{i \in \{1, \ldots, N\}} \left[ G_{t-1}(i) + \ln p(S_j | S_i) \right]
\]

// Store the link to predecessor node
\[
2) \quad \psi_t(j) = \arg \max_{i \in \{1, \ldots, N\}} \left[ G_{t-1}(i) + \ln p(S_i | S_j) \right].
\]

FOR \( j = 1, \ldots, N \) DO // Consider the termination links
\[
\ldots T = N + 1; \quad G_T(j) = G_{N}(j) + \ln p(\# | S_j)
\]

// Select the best final link
\[
G_{\text{BEST}} = \max_{j \in \{1, \ldots, N\}} G_T(j); \quad \ldots \quad Q_T = \arg \max_{j \in \{1, \ldots, k\}} G_T(j).
\]

// Backtracking along stored links - reconstruct the solution path \( Q_{t \ldots T} \)

FOR \( t = T-1, \ldots, 1 \) DO:
\[
Q_t = \psi_t(Q_{t+1})
\]

---

Baum-Welch training (1)

The parameters of a discrete HMM model, \( \lambda = (\Pi, A, B) \) can be estimated by the Baum-Welch training (an EM-like approach, that uses the “forward-backward” algorithm to obtain current forward and backward “messages”).

Given a discrete HMM model and an observation sequence \( O = (O_1, \ldots, O_T) \), the Baum-Welch training is an ML-estimation of the parameters \( \lambda \).

\[
P_{HMM}(\lambda) = \log P(O | \lambda) = \log \left( \sum_{q \in S^T} P(O, q | \lambda) \right)
\]

where \( q \) – a state sequence of length \( T \), corresponding to observation \( O \). The goal function:

\[
\hat{\lambda} = \arg \max_{\lambda} P_{HMM}(\lambda)
\]
„Forward-backward” algorithm

Let $N$ – be the number of states.
The “forward term” - the probability of generating a partial sequence and ending up in state with index $j$ at time $t$:

$$\alpha_t(j) = P(O_1 \cdots O_t, q_t = s_j \mid \lambda)$$

The “backward term” - the probability of generating the remainder of the sequence, from state with index $i$ at time $t$:

$$\beta_t(i) = P(O_{t+1} \cdots O_T, q_t = s_i \mid \lambda)$$

Thus the probability of visiting the state with index $j$ at frame $t$ for a complete observation sequence $O$ is the product of above probabilities:

$$\alpha_t(j) \cdot \beta_t(j) = P(O, q_t = s_j \mid \lambda)$$

$$\alpha_t(j) = b_j(O_t) \left( \sum_{i=1}^{N} \alpha_{t-1}(i) \cdot a_{ij} \right)$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)$$

„Baum-Welch” algorithm (1)

REPEAT

1) With current $\lambda = \{\pi_i, a_{ij}, b_{jk} \mid i,j=1,...,N; k=1,...,M\}$ and $t=1,...,T$; get $\alpha_t(j), \beta_t(i)$ by the "forward-backward" algorithm.

2) (The E-step) The expected probability of transition $s_i \rightarrow s_j$ at time $t$, given observation sequence $O=(O_1, \ldots, O_T)$ is:

$$\xi_t(i,j) = P(q_t = s_i, q_{t+1} = s_j \mid O, \lambda)$$

$$\xi_t(i,j) = \frac{P(q_t = s_i, q_{t+1} = s_j, O \mid \lambda)}{P(O \mid \lambda)} = \frac{\alpha_t(i) \cdot a_{ij} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_t(i) \cdot \beta_t(i)}$$

$$\gamma(i,j) - \text{the expected probability that the transition from state } i \text{ to state } j \text{ appears, at any time from 1 to } T: \gamma(i,j) = \sum_t \xi_t(i,j)$$
„Baum-Welch” (2)

\( \gamma(i) \) - the expected probability that state \( i \) is visited, at any time from 1 to \( T \):
\[
\gamma(i) = \sum \sum \xi_t(i, j)
\]
The expected probability that state \( i \) emits the symbol (class) \( Y_k \):
\[
\gamma(i, Y_k) = \sum \sum \xi_t(i, j)
\]
The expected probability of visiting state \( S_i \) at particular time \( t \) is:
\[
\gamma_t(i) = P(q_t = S_i \mid \mathbf{O}, \lambda) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j) \beta_t(j)} = \sum_{j=1}^{N} \xi_t(i, j)
\]

3) (The M-step)
In this step we can re-estimate the HMM parameter: vector \( \pi \), matrices \( A \) and \( B \), by taking simple ratios between above terms.

The update rules for the parameters of a discrete HMM model:
\[
\hat{\pi}_i = \gamma_1(i) = \frac{\alpha_1(i) \beta_1(i)}{\sum_{j=1}^{N} \alpha_1(j) \beta_1(j)} = \sum_{j=1}^{N} \xi_1(i, j)
\]
\[
\hat{a}_{ij} = \frac{\gamma(i, j)}{\gamma(i)} = \frac{\sum_{t=1}^{T} \xi_t(i, j)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\sum_{t=1}^{T} \alpha_t(i) \cdot a_{ij} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)}{\sum_{t=1}^{T} \alpha_t(i) \cdot \beta_t(i)}
\]
\[
\hat{b}_{jk} = \frac{\gamma(j, Y_k)}{\gamma(i)} = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot \chi(O_t \in Y_k)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\sum_{t=1}^{T} \alpha_t(j) \cdot \beta_t(j) \cdot \chi(O_t \in Y_k)}{\sum_{t=1}^{T} \alpha_t(j) \cdot \beta_t(j)}
\]

Where \( \chi(a \in Y) = 1 \) if the formula in brackets is satisfied or \( \chi(.) = 0 \) – otherwise.
3. Sentence recognition

Two recognition steps – a many-word recognition is performed first (using a 2-layer HMM model) and alternative word hypotheses are generated; then a sentence recognition is performed with an additional layer of HMM that seeks for a syntactically proper and semantically meaningful word sequence.

Word lattice

The word recognition stage generates 1-, 2- or 3-syllable-based (partially alternative) word hypotheses.

Example
N-grams

The language model provides prior probability of given word sequence $P(\text{words})$. For $n$ words $(s_1 \ldots s_n)$ we have:

$$P(s_1 \ldots s_n) = P(s_1)P(s_2 \mid s_1) \cdots P(s_n \mid s_1 \ldots s_{n-1}) = \prod_{i=1}^{n} P(s_i \mid s_1 \ldots s_{i-1})$$

In practice $n$ needs to be of limited length.

The bigram model – the probability of next word depends only on the direct predecessor word:

$$P(s_i \mid s_1 \ldots s_{i-1}) \approx P(s_i \mid s_{i-1})$$

$$P(s_1 \ldots s_n) = P(s_1)P(s_2 \mid s_1) \cdots P(s_n \mid s_{n-1}) = \prod_{i=1}^{n} P(s_i \mid s_{i-1})$$

The distribution $P(s_i \mid s_{i-1})$ can be learned by a counting the relative frequency of the pairs of words in a large learning set.

N-gram – if the $N$-th word depends on $N$-1 predecessor words.

Smoothing an N-gram

Katz’ smoothing for a three-gram model:

$$P^*(w_i \mid w_{i-1}, w_{i-2}) = \begin{cases} 
\frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})} , & r > k \\
\frac{d_r C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})} , & 0 < r \leq k \\
\alpha(w_{i-1}, w_{i-2}) P(w_i \mid w_{i-1}) , & r = 0 
\end{cases}$$

- Discount rate

$$d_r = \frac{r}{r - \frac{n_1}{1 - (k+1)n_{k+1}}} , \text{ for } r \in \{1,2,\ldots,k\}$$

- Good Turing estimate

$$r^*_r = (r + 1) \frac{n_{r+1}}{n_r}$$

- Number of samples: $r = C(w_{i-1}, w_i)$,
Token passing search

Token passing search – a traditional name for generation of hypothetic word sequences in speech recognition that includes the call of Viterbi search for individual word recognition.

Here we perform first the multiple word recognition process, which generates the Word lattice by repetitive calls to Viterbi search for next 1-, 2- or 3-syllable set of frames. Then a search for prospective paths in the Word lattice is separately performed. Originally „tokens” have marked the current ends of path for path extension with no central search tree management. Here:

- A uniform-cost search
- The N-gram model is applied for eventual path pruning
- All full paths are stored for sentence recognition.

HMM for sentence recognition

A HMM for sentence modelling, representing a stochastic syntax and a meaning (semantics):

1. predefined word meanings (semantic categories)
2. syntactic word roles assigned to categories

Ad 1. Every sentence is combined from parts containing atomic semantic information. For example, an atomic part may be:

- a question form (when, where, at what time),
- a time period (at eight a.m., afternoon, at evening),
- destination (Warszawa), etc.

Ad 2. Syntactic roles in a sentence: subject, predicate, object;
A syntax category:
noun, verb, conjunction, adjective, number, etc.
Example: HMM of sentence

The word dictionary contains 38 words in base form, like: from, to, train, hour, minute, day, when, Monday, today, etc. extended by the names of train destinations.

Some words may have different grammar forms, but this is already handled in the word recognition stage - converted into the base form.

The states of HMM represent following 11 meaning categories: question attribute, departure form, day, day-time, train attribute, train, from, to, departure city, destination city, end of sentence.

Every state can emit several words with specific probability.

The non-zero state transitions allow to accept different syntax sequences having the same meaning.
Thank you