Pattern Recognition

...A pattern is essentially an arrangement. It is characterized by the order of the elements of which it is made, rather than by the intrinsic nature of these elements.

Norbert Wiener

What is Pattern Recognition?

- Pattern recognition (PR) is the scientific discipline that concerns the description and classification (recognition) of patterns (objects)
- PR techniques are an important component of intelligent systems and are used for many application domains
  - Decision making
  - Object and pattern classification

What is Pattern Recognition
-Definitions from the literature

- "The assignment of a physical object or event to one of several pre-specified categories" –*Duda and Hart*
- "A problem of estimating density functions in a high-dimensional space and dividing the space into the regions of categories or classes" – *Fukunaga*
- "Given some examples of complex signals and the correct decisions for them, make decisions automatically for a stream of future examples" –*Ripley*
- "The science that concerns the description or classification (recognition) of measurements" –*Schalkoff*
- "The process of giving names ω to observations **x**", – *Schürmann*
- Pattern Recognition is concerned with answering the question "***What is this?***" –*Morse*

Pattern Recognition and Related Areas

- Image processing
- Video processing
- Speech/audio processing
- Natural Language processing
- Machine learning
- Neural networks
- Database engineering
- Bioinformatics
- Much more ….

Machine Perception

- Build a machine that can recognize patterns:
  - Speech recognition
  - Fingerprint identification
  - OCR (Optical Character Recognition)
  - DNA sequence identification

An "Toy" Example: Fish Classification

- "Sort incoming Fish on a conveyor according to species using optical sensing"

Species

Sea bass

Salmon
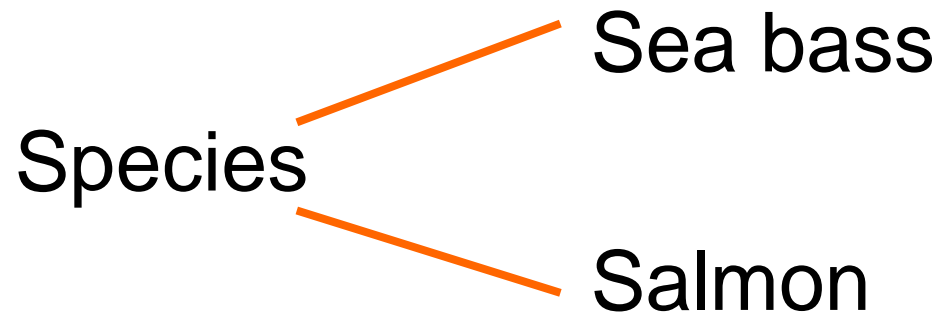
Problem Analysis

- Set up a camera and take some sample images to extract features
  - Length
  - Lightness
  - Width
  - Number and shape of fins
  - Position of the mouth, etc…
  - This is the set of all suggested features to explore for use in our classifier!
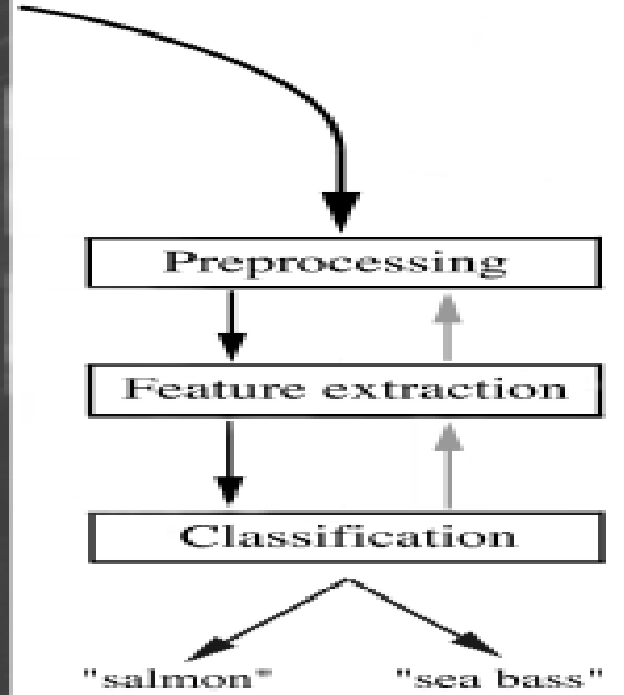
- Preprocessing
  - Use a segmentation operation to isolate fishes from one another and from the background
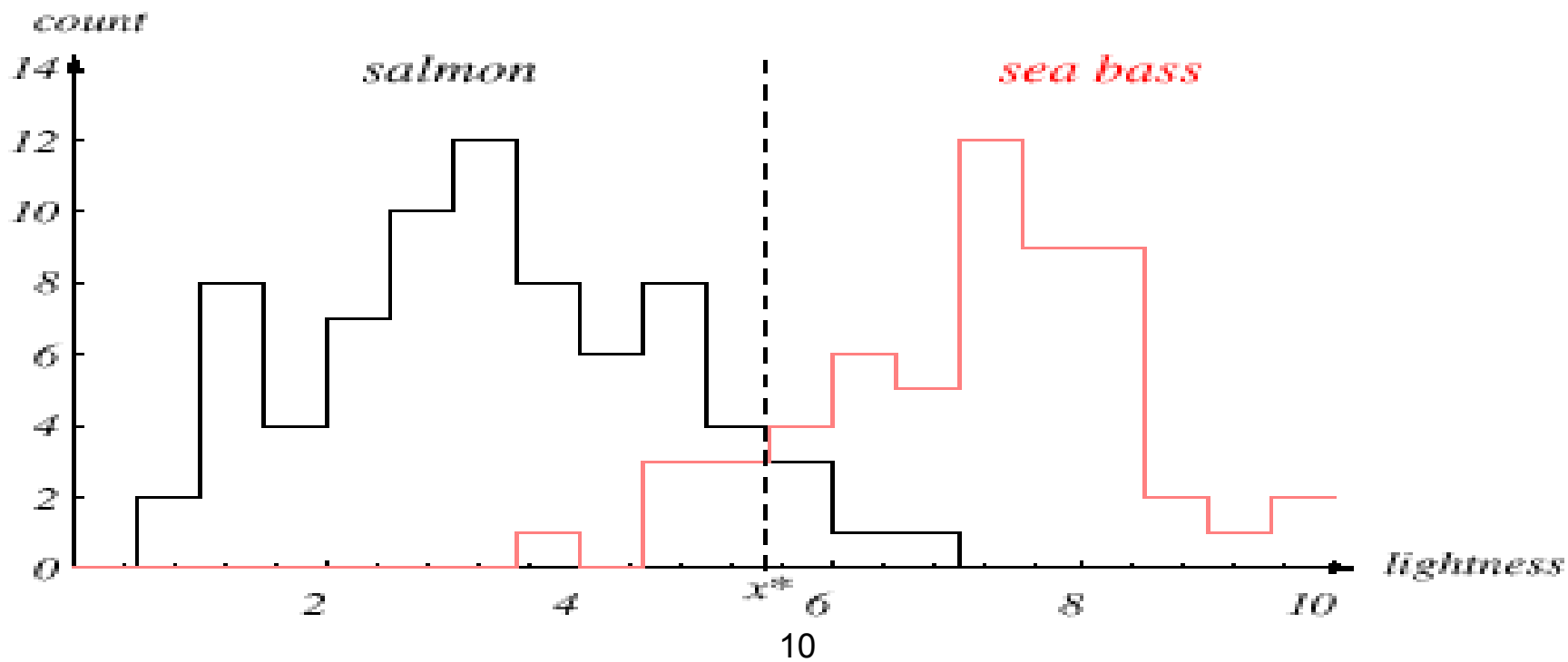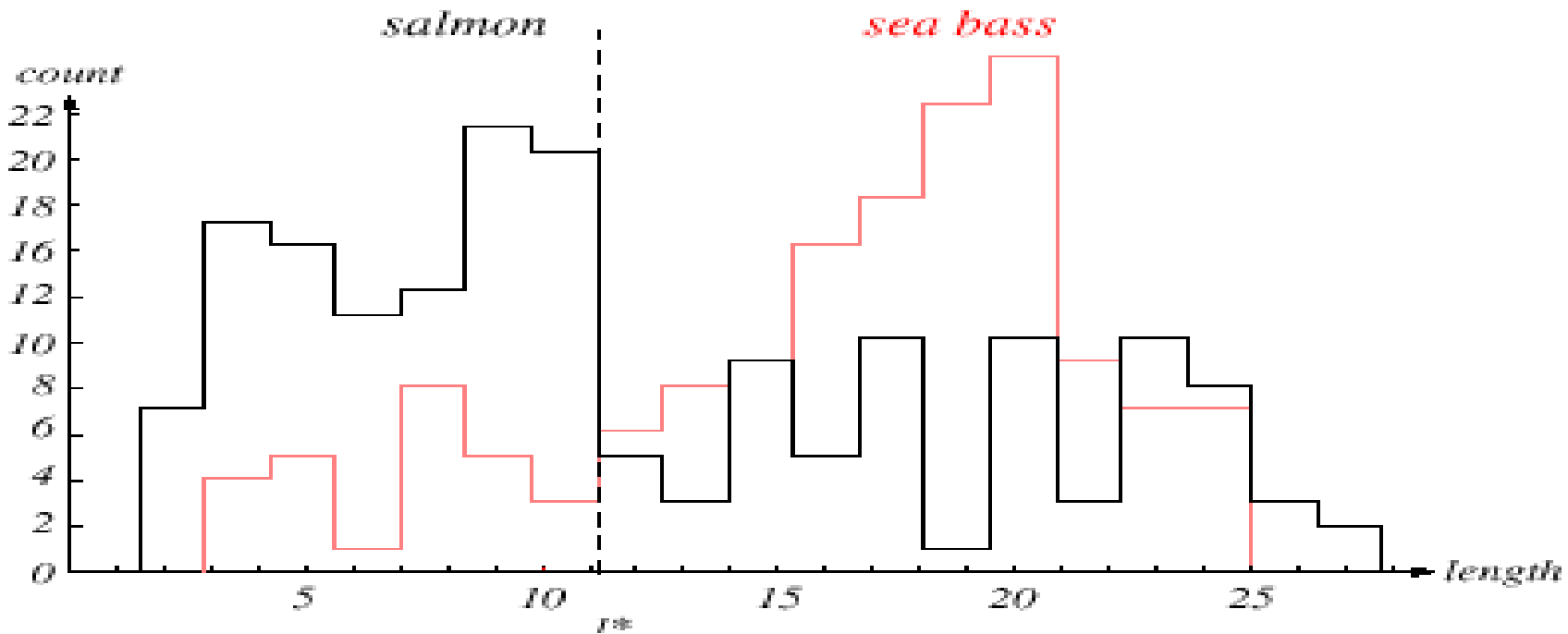 - To extract one fish for the next step
- Feature extraction
  - Measuring certain features of the fish to be classified
  - Is one of the most critical steps in the pattern recognition system design
- Classification
  - Select the length of the fish as a possible feature for discrimination

Preprocessing → Feature extraction → Classification → "salmon" / "sea bass"

salmon   sea bass

count

22
20
18
16
12
10
8
6
4
2
0

5   10   15   20   25   length

*r**

salmon   sea bass

count

14

12

10

8

6

4

2

0

2   4   *x** 6   8   10   lightness

10

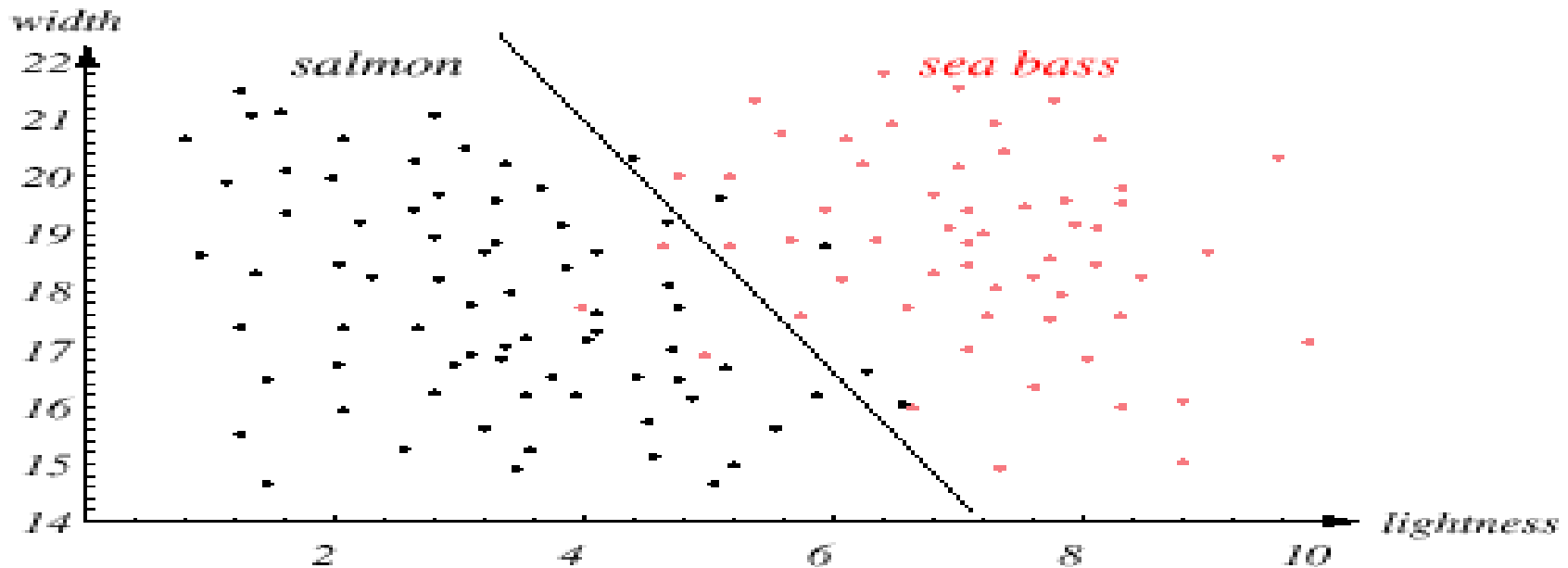# Adopt the lightness and add the width of the fish

Fish $\qquad$ $x^T = [x_1, x_2]$

Lightness $\qquad$ **Width**
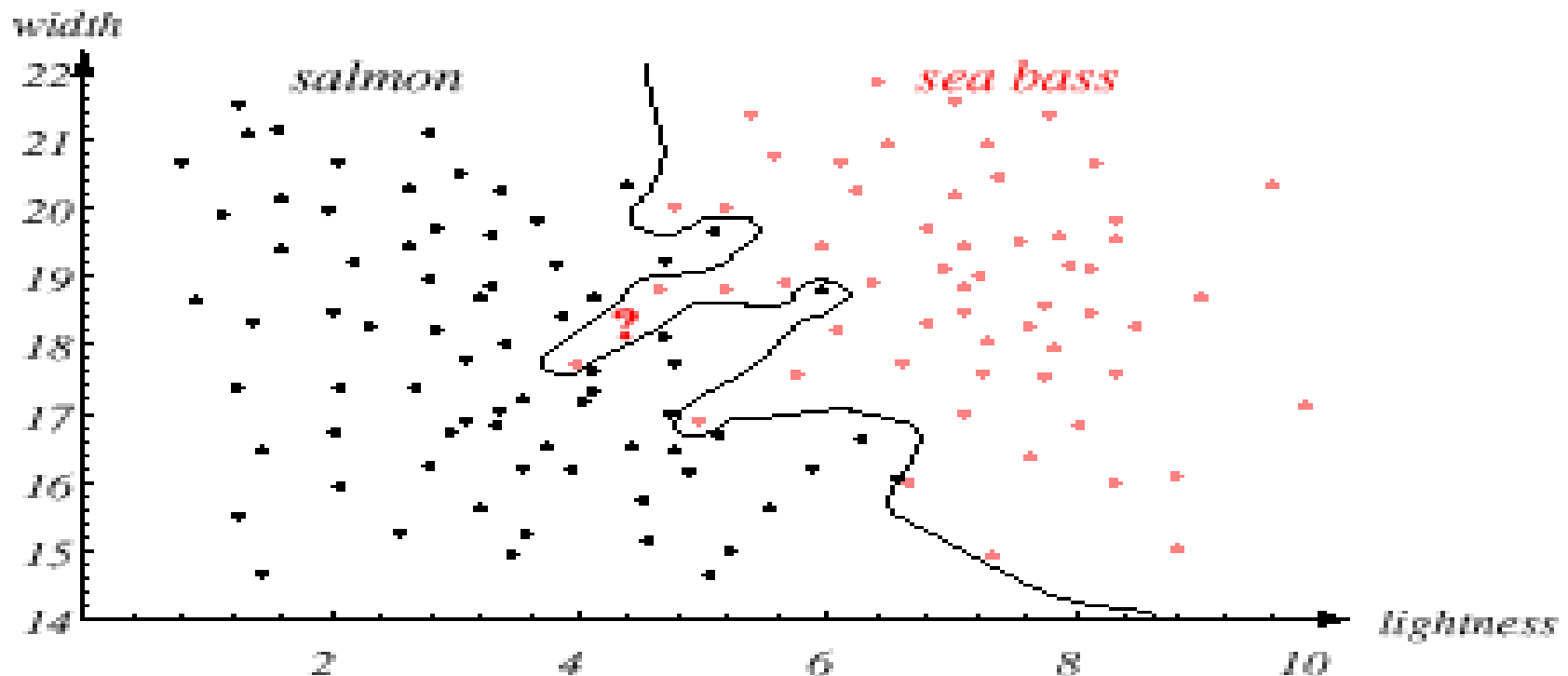
- We might add other features that are not correlated with the ones we already have. A precaution should be taken not to reduce the performance by adding such "noisy features"

- Ideally, the best decision boundary should be the one which provides an optimal performance such as in the following figure:

Pattern Recognition Systems

- ## Sensing
  - Use of a transducer (camera or microphone)
  - PR system depends on the bandwidth, the resolution sensitivity distortion of the transducer, etc.

- ## Segmentation and grouping
  - Patterns should be well separated and should not overlap

decision

post-processing ← costs

adjustments for context

classification ← adjustments for context

adjustments for missing features

feature extraction

segmentation

sensing

input
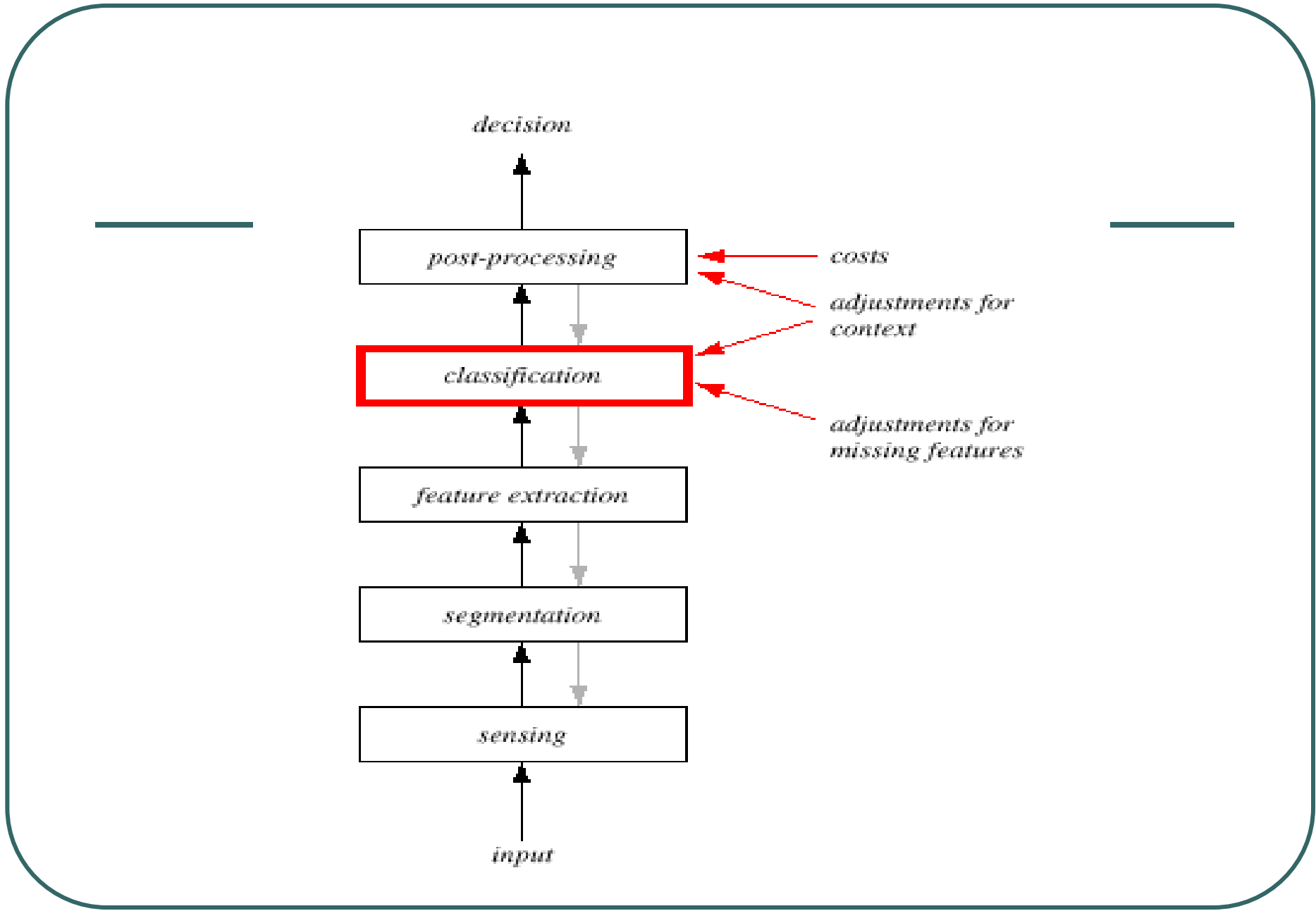
- # Feature extraction
  - Discriminative features
  - Invariant features with respect to translation, rotation and scale.

- # Classification
  - Use a feature vector provided by a feature extractor to assign the object to a category

- # Post Processing
  - Exploit context dependent information other than from the target pattern itself to improve performance

The Design Cycle

- Data collection
- Feature Choice
- Model Choice
- Training
- Evaluation
- Computational Complexity

start

collect data

choose features

choose model

prior knowledge
(e.g., invariances)

train classifier

evaluate classifier

end
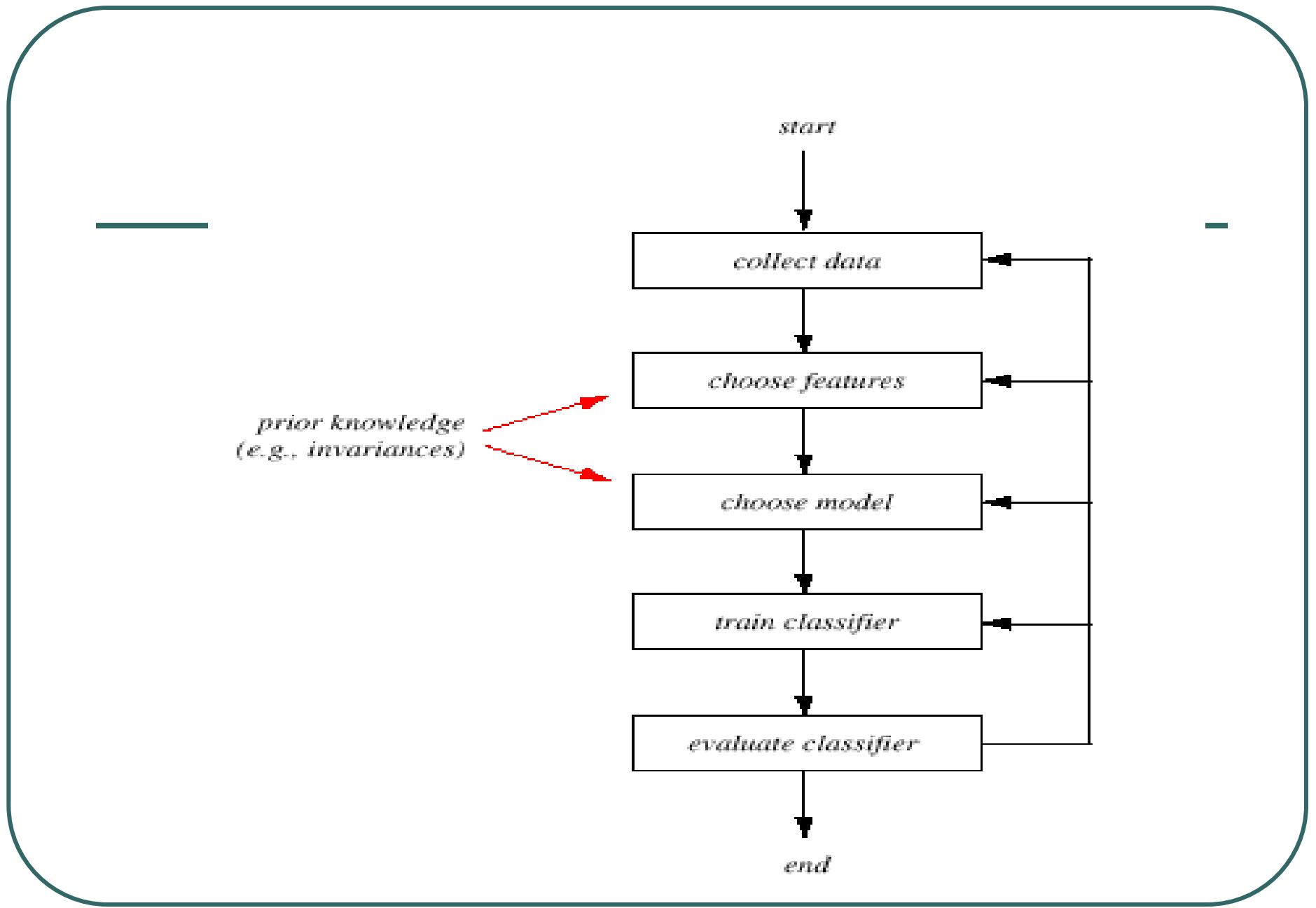
- ## Data Collection

  - How do we know when we have collected an adequately large and representative set of examples for training and testing the system?

- ## Feature Choice

  - Depends on the characteristics of the problem domain. Simple to extract, invariant to irrelevant transformation, insensitive to noise.

- ## Model Choice
  - Unsatisfied with the performance of our fish classifier and want to jump to another class of model

- ## Training
  - Use data to determine the classifier. Many different procedures for training classifiers and choosing models

- Evaluation
  - Measure the error rate for:
    - Different feature sets
    - Different training methods
    - Different training and test data sets
- Computational Complexity
  - What is the trade-off between computational ease and performance?
  - (How an algorithm scales as a function of the number of features, patterns or categories?)

Supervised & Unsupervised Learning

- ## Supervised learning
  - A teacher provides a category label or cost for each pattern in the training set (i.e., ground truth based on experts' knowledge)

- ## Unsupervised learning
  - The system forms clusters or "natural groupings" of the input patterns

- ## Semi-supervised learning
  - Use both labeled and un-labeled patterns to reduce the labeling cost

Approaches for PR

- **Statistical (StatPR)**

  Patterns classified based on an underlying statistical model of the features
  - The statistical model is defined by a family of **class-conditional probability** density functions $Pr(x|ci)$ (Probability of feature vector x given class ci)
- **Neural (NeurPR)**
  - Classification is based on the response of a network of processing units (neurons) to an input stimuli (pattern)
    - "Knowledge" is stored in the **connectivity and strength of the synaptic weights**
  - NeurPR is a trainable, non-algorithmic, black-box strategy
  - NeurPR is very attractive since
    - it requires minimum a priori knowledge
    - with enough layers and neurons, an ANN can create **any** complex decision region
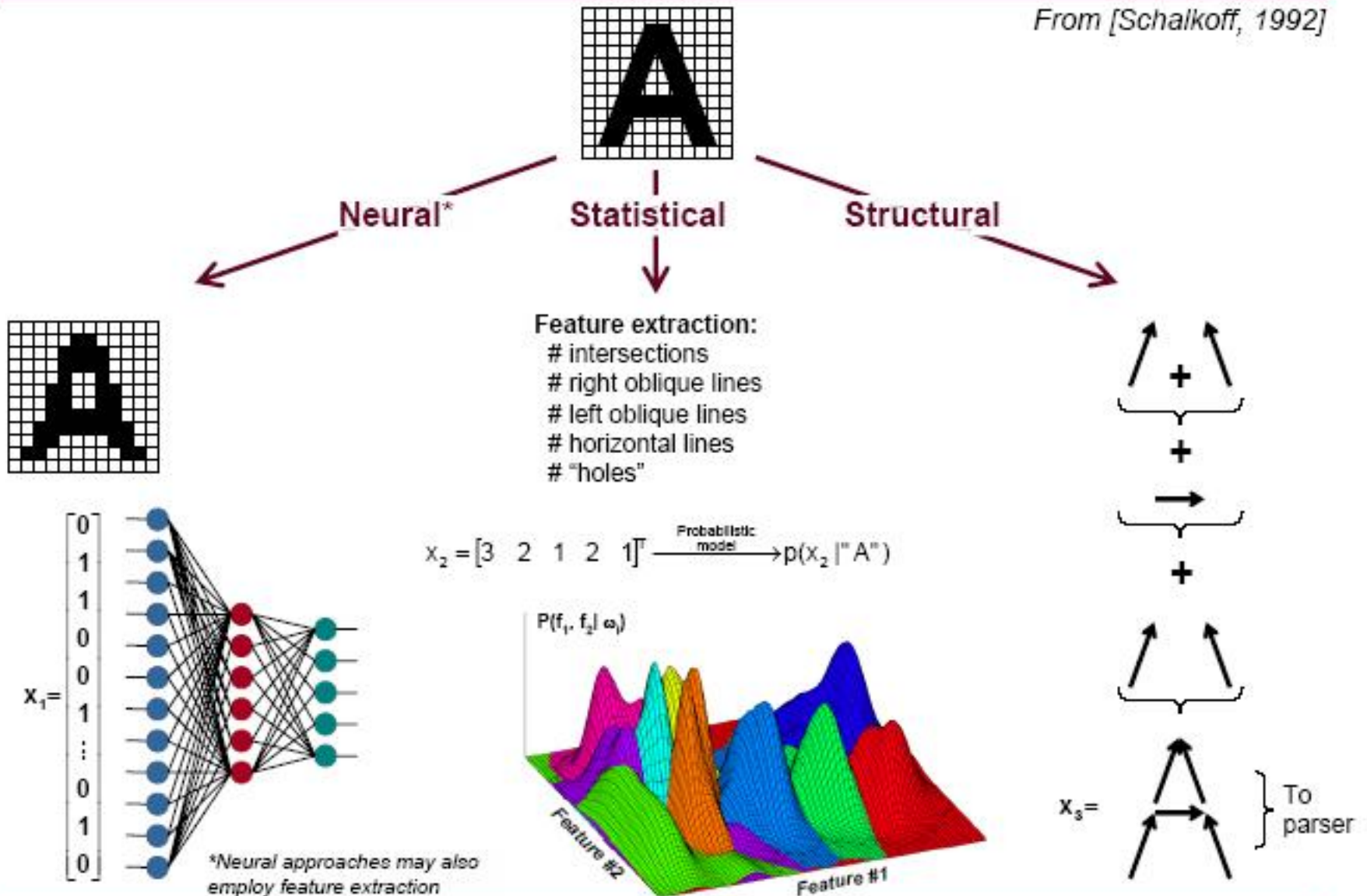- **Syntactic (SyntPR)**
  - Patterns classified based on measures of structural similarity
    - "Knowledge" is represented by means of **formal grammars or relational descriptions** (graphs)
  - SyntPR is used not only for classification, but also for description
    - Typically, SyntPR approaches formulate hierarchical descriptions of complex patterns built up from simpler sub patterns

Example: neural, statistical and structural OCR

Neural*   **Statistical**   **Structural**

Feature extraction:
- # intersections
- # right oblique lines
- # left oblique lines
- # horizontal lines
- # "holes"

$$x_2 = \begin{bmatrix} 3 & 2 & 1 & 2 & 1 \end{bmatrix}^T \xrightarrow{\text{Probabilistic model}} p(x_2 \mid "A")$$

$P(f_1, f_2 \mid \omega_i)$

Neural approaches may also employ feature extraction

$x_1 =$

Feature #2

Feature #1

$x_3 =$ } To parser

23

Decision-Theoretic Methods

- ### Decision (discriminant) function

Given pattern vector $\mathbf{x} = (x_1, x_2, ... x_n); \ \mathbf{x} \in R^n$

Given pattern classes $\omega_1, \omega_2, ... \omega_W$

Find W decision functions $d_1(\mathbf{x}), d_2(\mathbf{x}), ... d_W(\mathbf{x})$

such that, if $\mathbf{x} \in \omega_i$ then $d_i(\mathbf{x}) > d_j(\mathbf{x}) \ \ j = 1, 2, ... W; j \neq i$

- ### Decision boundary

$$d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0$$

$$\text{For two classes } \omega_i, \omega_j : \ d_{ij}(\mathbf{x}) = d_i(\mathbf{x}) - d_j(\mathbf{x}) \begin{cases} > 0 \ \mathbf{x} \in \omega_i \\ = 0 \quad ? \\ < 0 \ \mathbf{x} \in \omega_j \end{cases}$$

Decision-Theoretic Methods

- Matching: each class is represented by a protopype pattern vector. A predifined metric is needed
  - Minimum distance clasifier
  - Matching by correlation
- Optimal Statistical Classifiers
  - Bayes classifier
- Neural Networks
  - Perceptrons
  - Layers
  - Training

Matching

- Minimum distance classifier
  - Prototype def. as the mean vector of the class $\mathbf{m}_j = \dfrac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x}_j \quad j = 1,2,...,W$

  - Comparing the Eucledian distances $D_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_j\| \quad j = 1,2,...,W$

  - Decision function $d_j(\mathbf{x}) = \mathbf{x}^T \mathbf{m}_j - \dfrac{1}{2}\mathbf{m}_j^T \mathbf{m}_j \quad j = 1,2,...,W$

  - Decision boundary between two classes
    $$d_{ij}(\mathbf{x}) = d_i(\mathbf{x}) - d_j(\mathbf{x}) = \mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j) - \frac{1}{2}(\mathbf{m}_i - \mathbf{m}_j)^T(\mathbf{m}_i - \mathbf{m}_j) = 0$$

    - The surface is the perpendicular bisector of the line segment joining m_i and m_j. For n=2 it is a line, for n=3 it is a plane, and for n>3 it is a hyperplane.
    - Controlable mean separation and class spread

Matching

- Matching by correlation
  - Find matches of a subimage $w(x,y)$ of size JxK within the image $f(x,y)$ of size MxN.

$$c(x,y) = \sum_s \sum_t f(s,t)w(x+s),y+t) \quad x = 0,1,..M-1;\ y=0,1...N-1$$

  - a relative (normalized) *correlation coefficient* is prefered

$$\gamma(x,y) = \frac{\sum_s \sum_t \left[f(s,t)-\bar{f}(s,t)\right]\left[w(x+s),y+t)-\bar{w}\right]}{\left\{\sum_s \sum_t \left[f(s,t)-\bar{f}(s,t)\right]^2 \sum_s \sum_t \left[w(x+s),y+t)-\bar{w}\right]^2\right\}^{1/2}}$$

  - vulnerable to scale changes or rotation changes
  - The non-normalized version can be realized in FFT domain as well

Optimal Statistical Classifiers

- *Optimality* in the sence that the classifier yields the lowest probability of commiting classification error
  - Probability that $\mathbf{x}$ comes from ω_i: $p(\omega_i/\mathbf{x})$
  - If $\mathbf{x}$ is assigned (wrongly) to ω_j : loss L_{ij}
  - Conditional averaged risk (loss) $\quad r_j(\mathbf{x}) = \sum_{k=1}^{W} L_{kj}\, p(\omega_k/\mathbf{x})$

$$p(A/B) = [\,p(A)\,p(B/A)\,]/\,p(B)$$

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{k=1}^{W} L_{kj}\, p(\mathbf{x}/\omega_k) P(\omega_k); \text{ where } \begin{array}{l} p(\mathbf{x}/\omega_k) - \text{pdf of the patterns from class } \omega_k, \\ P(\omega_k) - \text{probabilty of occurence of class } \omega_k \end{array}$$

  - *Bayes classifier* minimises the cond. avaraged risk, i.e. assigns a new pattern $\mathbf{x}$ to the class ω_i if:

$$r_i(\mathbf{x}) < r_j(\mathbf{x}) \triangleright \sum_{k=1}^{W} L_{ki}\, p(\mathbf{x}/\omega_k) P(\omega_k) < \sum_{q=1}^{W} L_{qj}\, p(\mathbf{x}/\omega_q) P(\omega_q)$$

Bayes classifier (cont.)

The loss for correct decision is 0, the loss for any incorrect decision is 1. Then:

$L_{ij} = 1 - \delta_{ij}$, where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$.

$$r_j(\mathbf{x}) = \sum_{k=1}^{W}(1 - \delta_{kj})p(\mathbf{x}/\omega_k)P(\omega_k) = p(\mathbf{x}) - p(\mathbf{x}/\omega_j)P(\omega_j)$$

Decision : $\mathbf{x} \in \omega_i$ if $p(\mathbf{x}) - p(\mathbf{x}/\omega_i)P(\omega_i) < p(\mathbf{x}) - p(\mathbf{x}/\omega_j)P(\omega_j)$

Equavalent ly : $p(\mathbf{x}/\omega_i)P(\omega_i) > p(\mathbf{x}/\omega_j)P(\omega_j)$

Decision fuction : $d_j(\mathbf{x}) = p(\mathbf{x}/\omega_j)P(\omega_j)$ $j = 1,2,...W$

- Two probabilities needed. While $P(\omega_j)$ is easy to find (estimate), $p(\mathbf{x}/\omega_i)$ requires multivariate probability methods for its estimation. This is too complicated to be used in practice. Instead, analytical expressions (models) of the pdf are used. The necessary parameters are estimated from sample patterns from each class.

Bayes classifier (cont.)

- ## B.C. for Gaussian pattern classes
  - Consider 1-D case ($n=1$), two classes ($W=2$) governed by Gaussian densities $\mathcal{N}(m_1,\sigma_1)$, $\mathcal{N}(m_2,\sigma_2)$

$$d_j(x) = p(x/\omega_j)P(\omega_j) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{\frac{(x-m_j)^2}{2\sigma_j^2}} P(\omega_j) \quad j=1,2$$

  - For n-dimensional case, instead of simple variance, covariance matrix is involved

$$p(\mathbf{x}/\omega_j) = \frac{1}{(2\pi)^{n/2}|\mathbf{C}_j|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_j)^T(\mathbf{x}-\mathbf{m}_j)}$$

where $\mathbf{m}_j = E_j\{\mathbf{x}\};\ \ \mathbf{C}_j = E_j\{(\mathbf{x}-\mathbf{m}_j)^T(\mathbf{x}-\mathbf{m}_j)\};\ \ E_j\{.\} - \text{expected value}$

Approximations of $E_j\{.\}$: $\mathbf{m}_j = \frac{1}{N_j}\sum_{\mathbf{x}\in\omega_j}\mathbf{x};\ \ \mathbf{C}_j = \frac{1}{N_j}\sum_{\mathbf{x}\in\omega_j}\mathbf{x}\mathbf{x}^T - \mathbf{m}_j\mathbf{m}_j^T$

Bayes classifier (cont.)

- ## B.C. for Gaussian pattern classes
  - Exponents allow working with natural logarithms, since logarithm is a monotonically increasing function preserving the numerical order

  $$d_j(\mathbf{x}) = \ln p(\mathbf{x}/\omega_j) + \ln P(\omega_j)$$

  $$d_j(\mathbf{x}) = \ln P(\omega_j) - \frac{n}{2}\ln 2\pi - \frac{n}{2}\ln|\mathbf{C}_j| - \frac{1}{2}\left[(\mathbf{x}-\mathbf{m}_j)^T \mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j)\right]$$

  $$\text{or } d_j(\mathbf{x}) = \ln P(\omega_j) - \frac{n}{2}\ln|\mathbf{C}_j| - \frac{1}{2}\left[(\mathbf{x}-\mathbf{m}_j)^T \mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j)\right]$$

  - The decision functions in n-D space are *hyperquadrics (*quadratic function in n-D space*).*
  - Simplifications:
    - If all covariance matrices are equal => linear decision functions (*hyperplanes*)
    - $\mathbf{C}=\mathbf{I}$, $P(\omega_j)=1/W$. Then Bayes clsfr reduces to min. distance clsfr
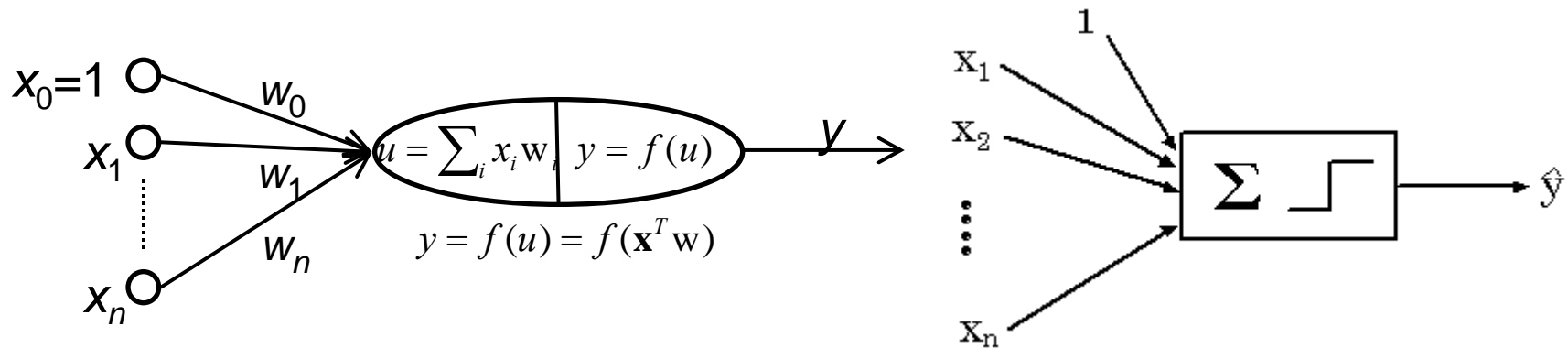
31

Neural Networks

- Preliminaries
  - *Training patterns* and *training sets*.
    - The process by which a training set is used to obtain decision functions is called *learning* or *training*

      e.g. the training is used to determine the parameters of the decision function (means, covariance matrices)
    - The statistical properties of pattern classes often are unknown (hard to estimate). Such problems are best handled by direct training (no need to make assumptions regarding the unknown pdfs)
  - *Neurons*: non-linear computing elements
    - Organized in *networks*
    - Learning mashines called *perceptrons*

Perceptron

- The appropriate weights are applied to the inputs, and the resulting weighted sum passed to a function which produces the output y

$$x_0 = 1 \quad w_0$$

$$x_1 \quad w_1$$

$$x_n \quad w_n$$

$$u = \sum_i x_i w_i \quad y = f(u)$$

$$y = f(u) = f(\mathbf{x}^T \mathbf{w})$$

$$X_1 \quad 1$$

$$X_2$$

$$\Sigma \quad \hat{y}$$

$$X_n$$

Neural Networks

- Perceptron for two pattern classes (see fig. above).
  - Weights modify the inputs to the input ot an activation function
  - Decision boundary is a hyperplane in n-D space
    - First n coeff. establish the orientation, while w_{n+1} determines the distance to the origin
  - Formally the free weight w_{n+1} can be included in the summation part by assuming one more input
  - The key problem is to find the weight vector $\mathbf{w}$ using a given training set of patterns from each of two classes

Neural Networks

- Training algorithms
  - Linearly separable classes: an iterative algorithm
    - $\mathbf{w}(1)$ –initial weight vector (arbitrary chosen)
    - at the k-th iterative step
      - if $\mathbf{y}(k) \in \omega_1$, and $\mathbf{w}^T(k)\mathbf{y}(k) \leq 0$, replace $\mathbf{w}(k)$ with
        $\mathbf{w}(k+1) = \mathbf{w}(k) + c\mathbf{y}(k)$, c is a positive correction increment
      - if $\mathbf{y}(k) \in \omega_2$, and $\mathbf{w}^T(k)\mathbf{y}(k) \geq 0$, replace $\mathbf{w}(k)$ with
        $\mathbf{w}(k+1) = \mathbf{w}(k) - c\mathbf{y}(k)$
      - otherwise, leave $\mathbf{w}(k)$ unchanged $\mathbf{w}(k+1) = \mathbf{w}(k)$
  - The algorithm converges if the two training sets are linearly separable.
    This is called the *perceptron training theorem*

Neural Networks

- Training for nonseparable classes: *delta rule*

Criterion function : $J(\mathbf{w}) = \dfrac{1}{2}(r - \mathbf{w}^T\mathbf{y})^2$, where $r = +1$, if $\mathbf{y} \in \omega_1$ and $r = -1$, if $\mathbf{y} \in \omega_2$

$J(\mathbf{w})$ has minimum when $r = \mathbf{w}^T\mathbf{y}$, hence adjust $\mathbf{w}$ in the direction of the negative gradient

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha\left[\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right]_{\mathbf{w}=\mathbf{w}(k)}$$

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -(r - \mathbf{w}^T\mathbf{y})\mathbf{y} \implies \mathbf{w}(k+1) = \mathbf{w}(k) + \alpha(r(k) - \mathbf{w}^T(k)\mathbf{y}(k))\mathbf{y}(k)$$

$\mathbf{w}(k+1) - \mathbf{w}(k) = \Delta\mathbf{w} = \alpha e(k)\mathbf{y}(k)$, where $e(k) = r(k) - \mathbf{w}^T(k)\mathbf{y}(k)$,

$e(k)$ is the error commited with $\mathbf{w}(k)$. If change to $\mathbf{w}(k+1)$ but leave the same pattern

$e(k) = r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k)$. The change of the error is

$\Delta e(k) = (r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k)) - (r(k) - \mathbf{w}^T(k)\mathbf{y}(k)) = -(\mathbf{w}^T(k+1) - \mathbf{w}^T(k))\mathbf{y}(k)$

$\Delta e(k) = -\Delta\mathbf{w}^T\mathbf{y}(k) = -\alpha e(k)\mathbf{y}^T(k)\mathbf{y}(k) = -\alpha e(k)\|\mathbf{y}(k)\|^2$

Neural Networks

- Delta rule comments:
  - Changing the weights reduces the error by a factor determined by α and energy of $\mathbf{y}$
  - The choise of α controls the stability and speed of convergence.
    - For stability $0 < \alpha < 1$.
    - Practical range $0.1 < \alpha < 1.0$
  - The algorithm converges over the patterns of the training set. For separable classes, the solution may or may not produce a separating hyperplane
  - Can be generalized to more than two classes and for non-linear decision functions

Neural Networks

- Multilayer feedforward NN
  - One output layer (Q) and several intermediate layers. Usually the first layer (A) has the dimension of the input vectors
  - Each neuron has similar structure as the perceptron model. The hard-limiting function has been replaced by soft-limiting 'sigmoid'. The smooth function is prefered because of the differentiability.

$$h_j(I_j) = \frac{1}{1 + e^{-(I_j + \theta_j)/\theta_0}}; \quad \theta_j \text{ controls the offset, } \theta_0 \text{ controls the shape}$$

  - The parameter θ_j plays the same role as w_{n+1} in the perceptron model and can be added as a weight to an additional input
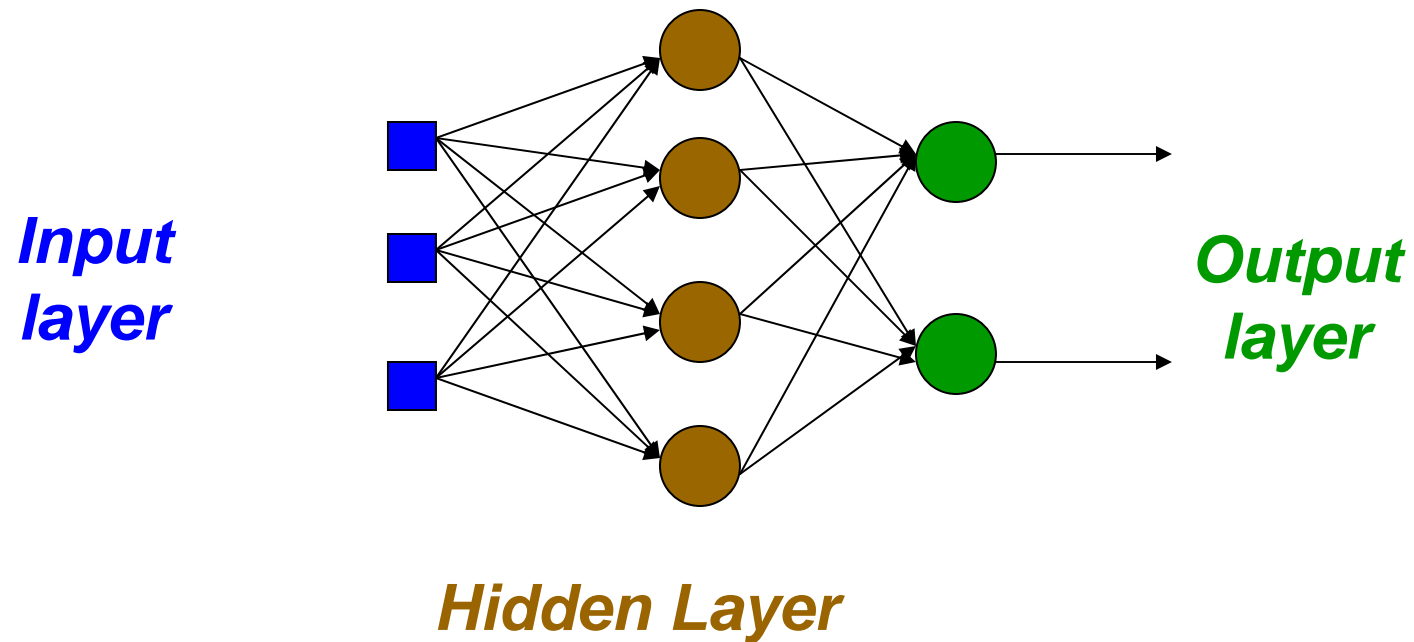 - Let K be the layer before the layer J. Then

$$I_j = \sum_{k=1}^{N_k} w_{jk} O_k \text{ for } j = 1,2...N_j$$

$$O_k = h_k(I_k) \text{ for } k = 1,2...N_k$$

  - Total of $N_j \times N_k$ coefficients are necessary to specify the weighting + N_j coeff. are needed to complete the nodes at layer J.

$$h_j(I_j) = \frac{1}{1 + e^{-(\sum_{k=1}^{N_k} w_{jk} O_k + \theta_j)/\theta_0}};$$

Multilayer Feedforward NN

Input layer

Hidden Layer

Output layer

FFNN NEURON MODEL

- The classical learning algorithm of FFNN is based on the gradient descent method. For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- A typical activation function that can be viewed as a continuous approximation of the step (threshold) function is the Sigmoid Function. A sigmoid function for node j is:

$$\varphi(\mathrm{v}_j) = \frac{1}{1+e^{-av_j}} \text{ with } a > 0$$

where $\mathrm{v}_j = \sum_i w_{ji} y_i$

with w_{ji} weight of link from node i

to node j and y_i output of node i

$\varphi(v_j)$

1—

Increasing a

v_j

-10 -8 -6 -4 -2   2   4   6   8   10
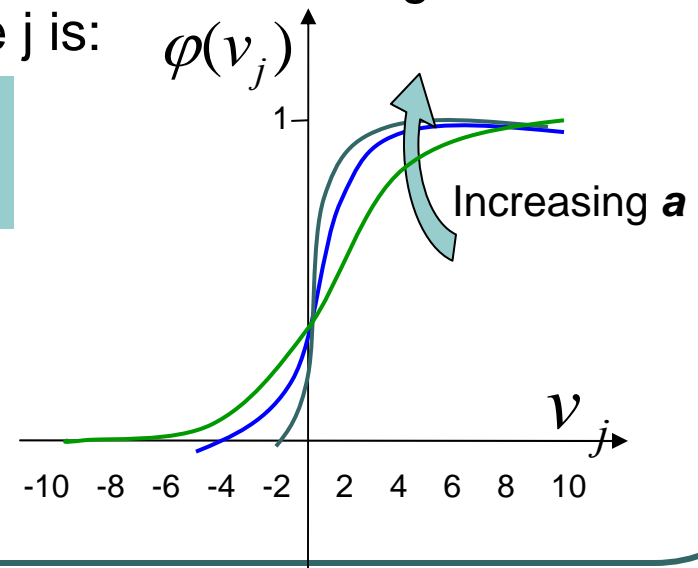
when a tends to infinity then φ tends to the step function

Multilayer NN

- Training by backpropagation

Total squared error for the output layer: $E_Q = \dfrac{1}{2}\sum_{q=1}^{N_q}(r_q - O_q)^2$

Adjusting the weights in proportion to the partial derivatives: $\Delta w_{qp} = -\alpha\dfrac{\partial E_Q}{\partial w_{qp}}$

By applying the chain rule: $\dfrac{\partial E_Q}{\partial w_{qp}} = \dfrac{\partial E_Q}{\partial I_q}\dfrac{\partial I_q}{\partial w_{qp}}$

$\dfrac{\partial I_q}{\partial w_{qp}} = \dfrac{\partial}{\partial w_{qp}}\sum_{p=1}^{N_p} w_{qp}O_p = O_p \Rightarrow \Delta w_{qp} = -\alpha\dfrac{\partial E_Q}{\partial I_q}O_p = -\alpha\delta_p O_p$, where $\delta_p = -\dfrac{\partial E_Q}{\partial I_q}$

How to compute $\dfrac{\partial E_Q}{\partial I_q}$? $\quad \delta_p = -\dfrac{\partial E_Q}{\partial I_q} = -\dfrac{\partial E_Q}{\partial O_q}\dfrac{\partial O_q}{\partial I_q}$

$\dfrac{\partial E_Q}{\partial O_q} = -(r_q - O_q);\quad \dfrac{\partial O_q}{\partial I_q} = \dfrac{\partial}{\partial I_q}h_q(I_q) = h_q{}'(I_q) \Rightarrow \delta_p = (r_q - O_q)h_q{}'(I_q)$

Finally: $\Delta w_{qp} = \alpha(r_q - O_q)h_q{}'(I_q)O_p = \alpha\delta_q O_p$

Multilayer NN

- Training by backpropagation: What happens in layer P ?

$$\Delta w_{qj} = \alpha (r_p - O_p) h_p'(I_p) O_j = \alpha \delta_p O_j$$

where the error term is: $\delta_p = (r_p - O_p) h_p'(I_p)$.

All terms (except r_p) are known or can be observed in the network.

How to restate δ_p in terms (quantities) that are known (observeble)?

$$\delta_p = -\frac{\partial E_p}{\partial I_p} = -\frac{\partial E_p}{\partial O_p} \frac{\partial O_p}{\partial I_p}; \quad \frac{\partial O_p}{\partial I_p} = \frac{\partial h_p(I_p)}{\partial I_p} = h_p'(I_p); \quad r_p \sim \frac{\partial E_p}{\partial O_p} = ?$$

$$-\frac{\partial E_p}{\partial O_p} = -\sum_{q=1}^{N_Q} \frac{\partial E_p}{\partial I_q} \frac{\partial I_q}{\partial O_p} = \sum_{q=1}^{N_Q} \left(-\frac{\partial E_p}{\partial I_q} \right) \frac{\partial}{\partial O_p} \sum_{p=1}^{N_p} w_{qp} O_p = \sum_{q=1}^{N_q} \left(-\frac{\partial E_p}{\partial I_q} \right) w_{qp} = \sum_{q=1}^{N_q} \delta_q w_{qp}$$

$$\delta_p = h_p'(I_p) \sum_{q=1}^{N_q} \delta_q w_{qp}$$

Multilayer NN

- Training by backpropagation: Summarise the procedure
  - For any layers K and J (K precedes J) compute the weights w_{jk}, which modify the connections between these two layers, by using
    $$\Delta w_{jk} = \alpha \delta_j O_k$$
  - If J is the output layer, δ_j is $\quad \delta_j = (r_j - O_j)h_j{}'(I_q)$
  - If J is an internal layer and P is the next layer (to the right), δ_j is

    $$\delta_j = h_j{}'(I_j)\sum_{p=1}^{N_p} \delta_p w_{jp} \text{ for } j = 1,2,...N_j.$$

  - Using an activation function with $\theta_0 = 1$ yelds

    $$h_j{}'(I_j) = O_j(1 - O_j)$$
    $$\delta_j = (r_j - O_j)O_j(1 - O_j) \text{ for the output layer}$$
    $$\delta_j = O_j(1 - O_j)\sum_{p=1}^{N_p} \delta_p w_{jp} \text{ for the internal layers}$$

Recent Advances in Pattern Recognition

- New applications
- Standard databases for performance benchmark of algorithms and systems
- Fusion methods to improve performance
  - Sensors, features, classifiers
- Robustness of algorithms and systems
- Utilizing contexts or expert information
- Many successful deployments
  - Speech recognition, handwritten character recognition, face detection, fingerprint recognition, automatic vehicle guidance, visual inspections, computer aided diagnosis for mammogram

Summary

- Pattern recognition systems aim to recognize patterns based on their features

- Classification is an important step in pattern recognition systems

- Pattern recognition algorithms and systems have been widely used in many application domains

- Challenges remain to achieve human like performance

- More in **SGN-2556 Pattern Recognition, 5th term**