Tfy-99.275 lecture 7

Pattern Recognition,
Classification techniques

Problem Statement

- In many applications we measure many variables from different subjects with the aim of building some useful input-output system.
- We often start with whatever variables that may be expected to be useful and then try to figure out how that variable set (or part of it) can be associated with known outputs.
- This involves issues like feature extraction, feature selection and classification performance evaluation

Feature Extraction

- reduces the amount of raw data while maintaining (or enhancing) the vital information for the pattern recognition task at hand
- features can range from low-level features (typically measurements, e.g., numerical values like signal amplitudes) to high-level features (e.g., 'symptom' feature for classification into 'diseases' classes)

Feature Selection

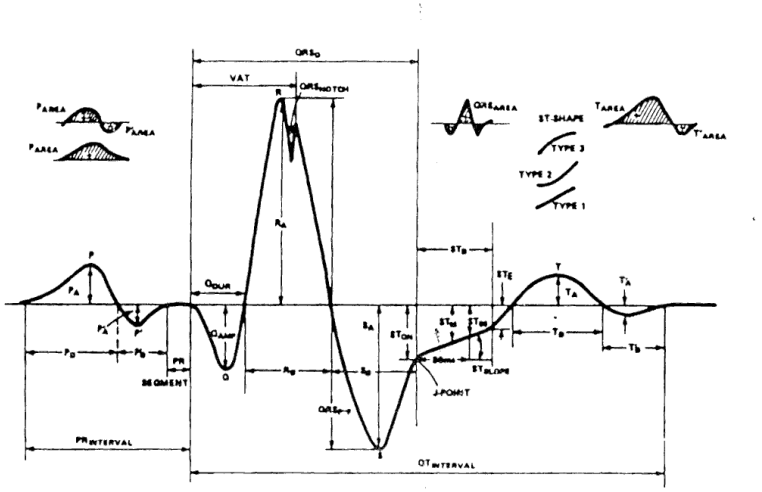
- reduce the set of data even further; select those features that are most relevant to the pattern recognition task at hand(for example, using principal component analysis)

Some examples of features

Features can be obtained from different ways of processing the signals

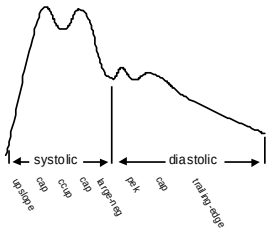
measurements	**results from transforms**	**underlying structure**
amplitude	polynomials	peaks
bias	Fourier	slopes
phase	cosine	corners
energy	wavelets	parametric models
moments		

Examples of the many features that can be obtained from a 'simple' ECG curve

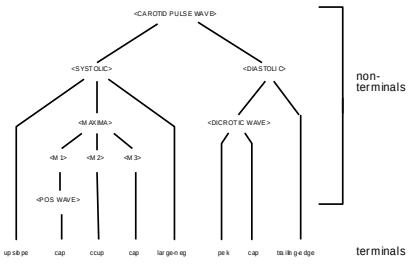


a structural description of a blood pressure waveform

a



b

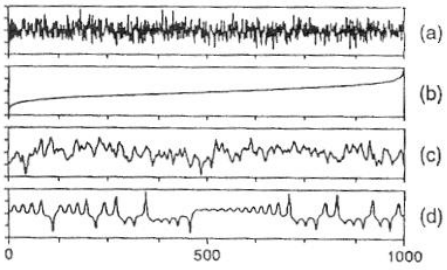


Feature Vectors

- comprise all features used to describe a pattern; it is a reduced-dimensional representation of that pattern
- the set of all features that could be used to describe a pattern is limited to those actually stated in the feature vector.

Watch out!

- A feature vector usually can not replace the original signal on a one-to-one basis; keep backups of your raw data at all times!



Example of four synthesized time series with identical means, standard deviations, and ranges. Series (c) and (d) also have identical autocorrelation functions and therefore identical power spectra.

Classification (pattern recognition)

- Identify to which class, *C*, an observation (pattern), x, from which the selected-feature vector was calculated, belongs.
- Patterns that are quantitatively similar may belong to the same class, in such cases clustering techniques or matching of patterns against prototypes can be used.
- However, also patterns that bear no obvious similarities to one another can also belong to the same class.

Pattern Recognition Techniques

- The huge amount of pattern recognition techniques is being used in biomedical applications can be (roughly) fitted into 3 main groups of techniques

techniques		**principle of operation**
statistical		classify upon probability distribution of characteristic measurement
syntactic		use a structural description of the shape of the signals, use interrelationships between features rather than values
artificial intelligence (AI)	symbolic AI	use symbolic representations of class structures, manipulate information by logic statements (expert systems)
	connectionist AI	relate characteristic feature sets to class structures using examples (neural networks)

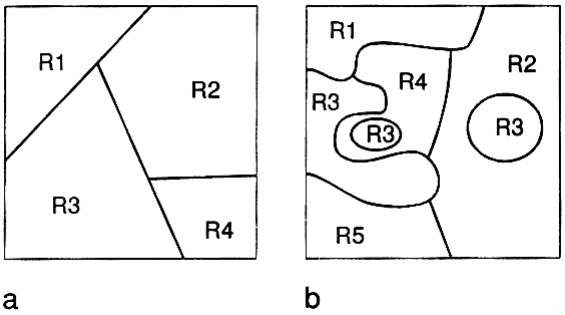
Statistical Pattern Recognition Techniques

- A set of *N* features extracted from a pattern can be treated as an *N*-dimensional vector ***x*** in an *N*-dimensional space. Such a vector is called a *feature vector* and the *N*-dimensional space is called *feature space*. Usually, the feature space is $\mathbf{R}^N$, but it can also be a subspace of $\mathbf{R}^N$.
- Classifications are based on the determination of *decision regions*. The feature space is thus divided into *decision regions* that are associated with classes.
- The borders between decision regions are called *decision boundaries.*

Decision Regions

- The classification is performed by determining the decision region in which the applied feature vector falls. The main problem in constructing a good classifier is the method of determination of the decision boundaries.

Examples of decision regions and boundaries in a 2-dimensional feature space:
a: decision regions separated by linear decision boundaries;
b: a general division of feature space into decision regions.



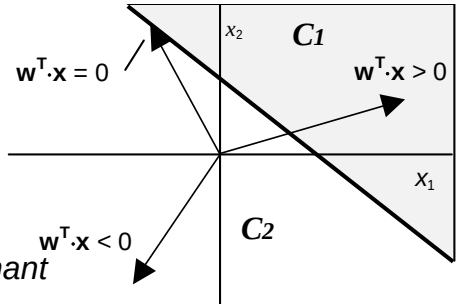
Linear Separation of Decision Regions

n-dimensional feature vector **x** with x_0 set to1 and a weight vector, **w**

$$y = w_0 \cdot 1 + w_1 \cdot x_1 + \mathrm{L} + w_n \cdot x_n$$

2-class separation problem

$$d(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \cdot \mathbf{x} \begin{cases} > 0 \text{ if } \mathbf{x} \in C_1 \\ \leq 0 \text{ if } \mathbf{x} \in C_2 \end{cases}$$



For an M-class separation problem we'll need M-1 such *linear discriminant functions*

Decision Functions*

In the general case the decision region to which a feature vector belongs is determined by a *decision function* (which is not necessary linear). A decision function $g_i(\boldsymbol{x})$ is defined for each decision region *i*. Often, the feature vector ***x*** is assigned to region *m* for which the relationship

$$g_m(\boldsymbol{x}) > g_i(\boldsymbol{x}) \; \forall_{i \neq m}$$

holds.

The linear discriminant function of the previous slide is a special (but popular) choice.

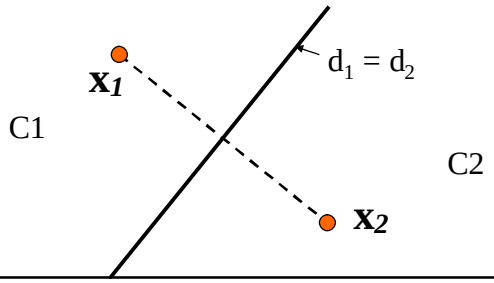
*Often, also the term 'Discriminant Functions' is used.

Minimum Distance Classifier

An example of a feature space with two decision regions their decision functions being:

$$d_1(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_1| \text{ and } d_2(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_2|.$$

In this case the feature vector is assigned to the decision region with the ***smallest*** decision function; this is a minimum distance classifier.



k-Nearest Neighbour Classifiers

Suppose we have *N* sample patterns {$\mathbf{s}_1$, $\mathbf{s}_2$, ..., $\mathbf{s}_N$} that we know belong to one of *M* classes {$\boldsymbol{C}_1$, $\boldsymbol{C}_2$, ..., $\boldsymbol{C}_M$}.

For a new, to classify, pattern **x**, we calculate the distance D($\mathbf{s}_i$,**x**) and assign **x** to the class of the sample pattern that is closest to **x** (the 'nearest neighbour').

This is not a very stable way of classifying as the classification is made upon the basis of one single sample pattern (which may be an outlier).

A more reliable estimate may be obtained by classifying using several sample patterns; *k* nearest neighbours and then use a majority vote. This is the *k-nearest-neighbour (k-NN) rule*.

Estimation of Discriminant Functions

The crucial point is to create these decision regions and discriminant functions. Two possible methods;

1. convert the so-called *a priori* class probability $P(c_i)$ into an *a posteriori* (based on measured patterns) probability $P(c_i|\boldsymbol{x})$;
2. minimize the expected error the classifier will make.

There are many methods available two those two approaches, it is beyond the scope of this course however to provide an in-depth review of them.

Clustering Methods

- Sometimes we don't want (or cannot) assign classes beforehand. We may then examine possible formation of inherent groups or clusters of the data in feature space and (possibly) label the clusters with class labels.
- This may be relatively easy in 2D, but for higher-dimensionial feature spaces this is a difficult task.
- We have to group the feature vectors on the basis of similarity, dissimilarity or distance measures.
- Examples of commonly used measures are the Euclidean distance, the Mahalanobis distance, and the normalised dot product (there exist many more)

Distance measures

- Euclidean distance, between feature vectors x and z

$$D_E^2 = \|\mathbf{x}-\mathbf{z}\|^2 = (\mathbf{x}-\mathbf{z})^T(\mathbf{x}-\mathbf{z}) = \sum_{i=1}^{n}(x_i - z_i)^2$$

- Normalised dot product $D_d = \dfrac{\mathbf{x}^T\mathbf{z}}{\|\mathbf{x}\|\|\mathbf{z}\|}$
- Mahalanobis distance $D_M^2 = (\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})$

  x is a feature vector compared to a pattern class with m as mean class vector and C as covariance matrix

$$\mathbf{C} = E\left[(\mathbf{y}-\mathbf{m})(\mathbf{y}-\mathbf{m})^T\right]$$

  The expectation is calculated over all vectors y that belong to the class. C provides covariance of all possible pairs of features. The diagonal contains variance of individual features. The matrix represents the 'scatter' of features belonging to a certain class.

Example of a clustering algorithm, the K-means algorithm

- goal: (iteratively) minimise sum of squared distances from all points in a cluster domain to the cluster centre.

1. choose K initial cluster centres, $z_1(1)$, $z_2(1)$..., $z_K(1)$ (the index in parenthesis indicates iteration number)
2. at iteration k distribute samples {x} among the K clusters using

$$\mathbf{x} \in S_j(k) \text{ if } \|\mathbf{x}-\mathbf{z}_j(k)\| < \|\mathbf{x}-\mathbf{z}_i(k)\| \;\forall\; i=1,2,...,K,\; i \neq j$$

3. from results of 2. calculate new cluster centres $z_j(k+1)$ so that the sum of squared distances from all points in $S_j(k)$ to the new cluster centre is minimised. The $z_j(k+1)$ that minimises this distance is simply the sample mean of $S_j(k)$, so

$$\mathbf{z}_j(k+1) = \frac{1}{N_j}\sum_{\mathbf{x}\in S_j(k)}\mathbf{x}, \quad j=1,2,...,K.$$

4. if $z_j(k+1) = z_j(k)$ for j=1,2,...K the algorithm has converged: terminate the iteration, otherwise go to step 2.

Note: one has to define the number of clusters and initial cluster centres beforehand.

Syntactic Pattern Recognition Techniques

- These techniques concentrate on the structure of the presented patterns rather than on assumed distributions of feature sets.
- Basic tenet: patterns belonging to the same class can be characterised by a common structural description.
- Descriptions are made using hierarchies of features. Low-level features are called *primitives* (e.g., arcs, corners, straight lines)
- Signals are decomposed into primitives by using *grammars*. A mechanism (the *parser*) matches the decomposed structural description against known structures and recognises the patterns.

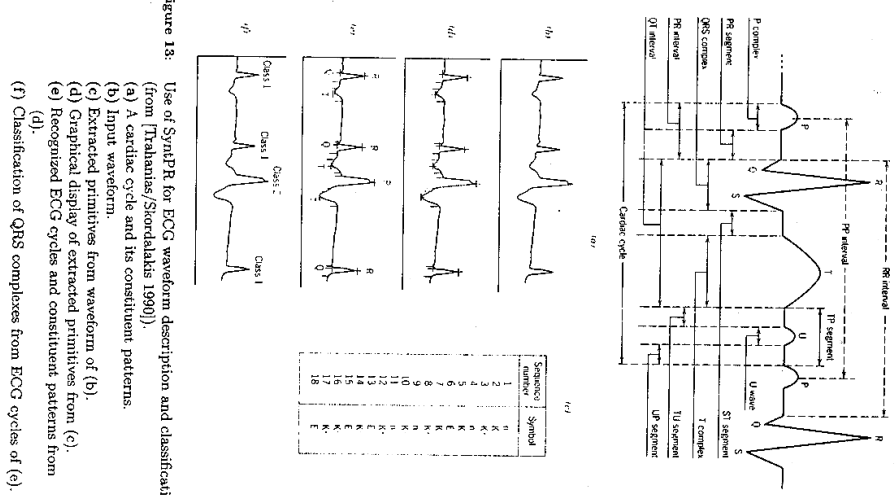


Figure 13: Use of SynPR for ECG waveform description and classification (from [Trahanias/Skordalakis 1990]).
(a) A cardiac cycle and its constituent patterns.
(b) Input waveform.
(c) Extracted primitives from waveform of (b).
(d) Graphical display of extracted primitives from (c).
(e) Recognized ECG cycles and constituent patterns from (d).
(f) Classification of QRS complexes from ECG cycles of (e).

Syntactic Pattern Recognition

- The usefulness of these syntactic techniques is sometimes believed to be limited for practical applications; parsing difficulties occur especially for noisy signals.
- There are some fields where they seem to be more effective than statistical approaches though. Example application areas include ECG and EP processing.

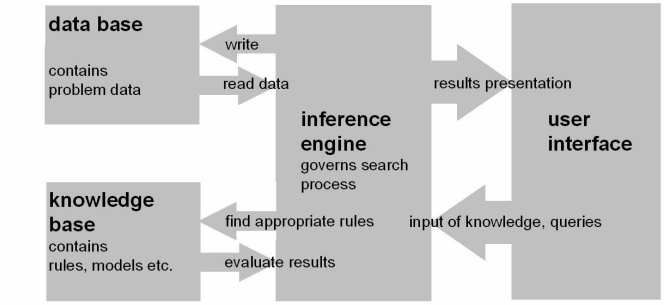
Artificial Intelligence (AI) Techniques

- These techniques concentrate mainly on two concepts:
  - *knowledge presentation* - the choice of a method to represent facts;
  - *learning* - the AI-based system should be able to increase its efficiency and effectivity when a similar or identical task is processed repeatedly.

Symbolic AI Techniques (Expert Systems)

- *symbolic representations* (or simply: symbols) of class structures are handled in such a way that the system is able to describe and identify the structures and states referred to by the symbols.
- For example, a symbol can be the word *square* to represent a square-shaped part of an image, or the words *low temperature* to represent a temperature in the range of -5 ... +5 °C. Usually, these symbols are manipulated by using logical statements (e.g. an `if...then...else...` construction).

General Structure of an Expert System



The *inference engine* searches the *knowledge base* (which contains the domain-specific knowledge) during a problem-solving task and also evaluates the various rules in the knowledge base to add results to the *data base*. The data base contains the current problem data.

Expert Systems

- The inference engine can use different approaches in its search process, for example *backward chaining* (start with the goal of the problem and repeatedly infer subgoals on the way to the final solution of the problem), or *forward chaining* (start with inferring the immediate consequences of the data and use consequences to come to further consequences, repeat until the final goal of the problem has been reached).
- *Expert systems* use knowledge of the problem domain to interpret data and to provide feedback to the user of the system, i.e. they can 'explain' how they interpreted the data.
- This domain knowledge is put into the expert system as a result of the interaction between a domain expert and a knowledge engineer: an AI specialist skilled in analysing the expert's problem-solving processes and encoding them in a computer system.

- Expert systems operate in a transparent manner, i.e., the path to their conclusions can be traced. The systems can explain or justify their conclusions, and they also enable developers to check on the operation of the systems themselves.
- One essential step in the construction of an expert system is the storage of knowledge in the knowledge base. To this end, this knowledge has to be present in a structured form. The transition of the domain expert's knowledge to rules cannot always be made; this is especially the case when this knowledge is partially based on experience. Another problem may occur when the domain expert's opinion or method of acting changes over time; it is then often difficult to change the knowledge base in order to reflect the altered situation.

Connectionist AI techniques

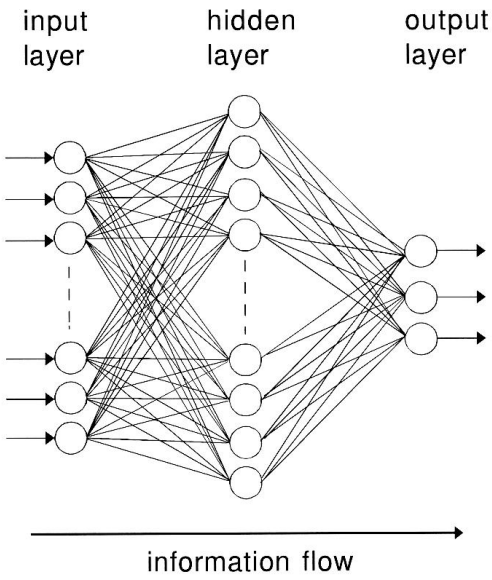
- This mainly concerns the use of artificial neural networks (ANNs).
- They are especially useful in finding solutions for problems where no, or very little, explicit knowledge is available and where sufficient representative specimen cases from which heuristic knowledge can be derived are available.

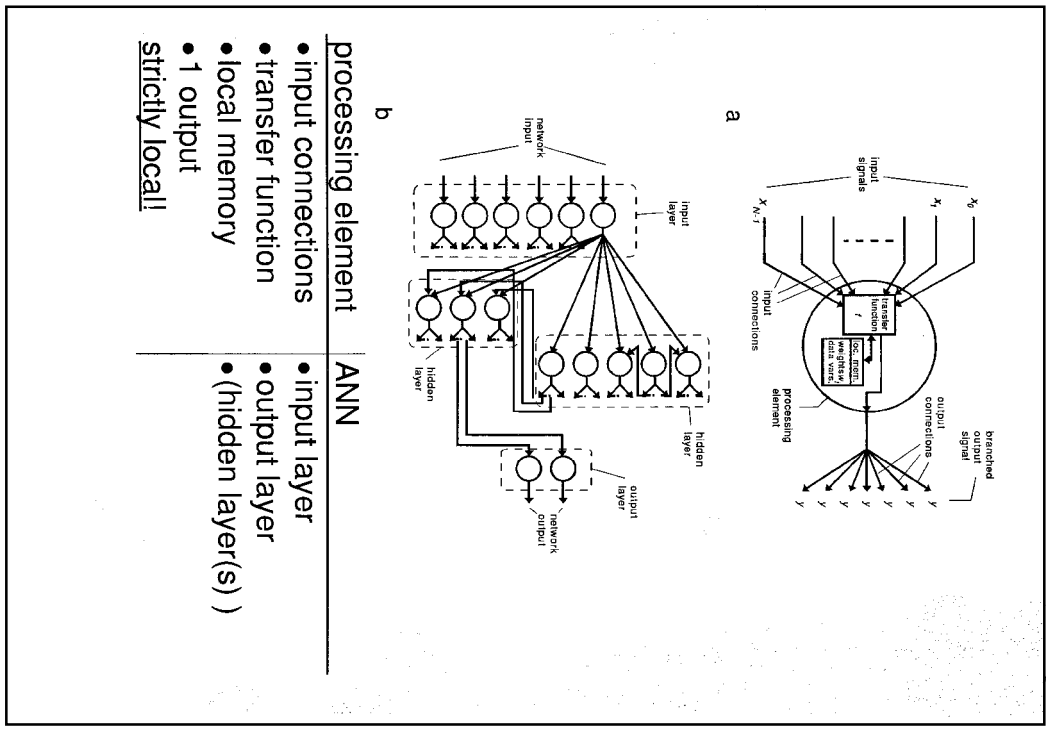
Programmed vs. Neurocomputing

- programmed computing: success dependent on the possibility to convert the solution to a given problem into a set of procedures or rules
- neurocomputing: success dependent on the existence of enough usable 'examples' of aspects of the problem

Artificial Neural Networks

- a *large* number of *simple* (*non-linear*) units that are *densely interconnected*
- information is stored in *weights* that are associated with the *interconnections*
- the network 'learns' by *adapting* the network *weights* in response to information present in its environment



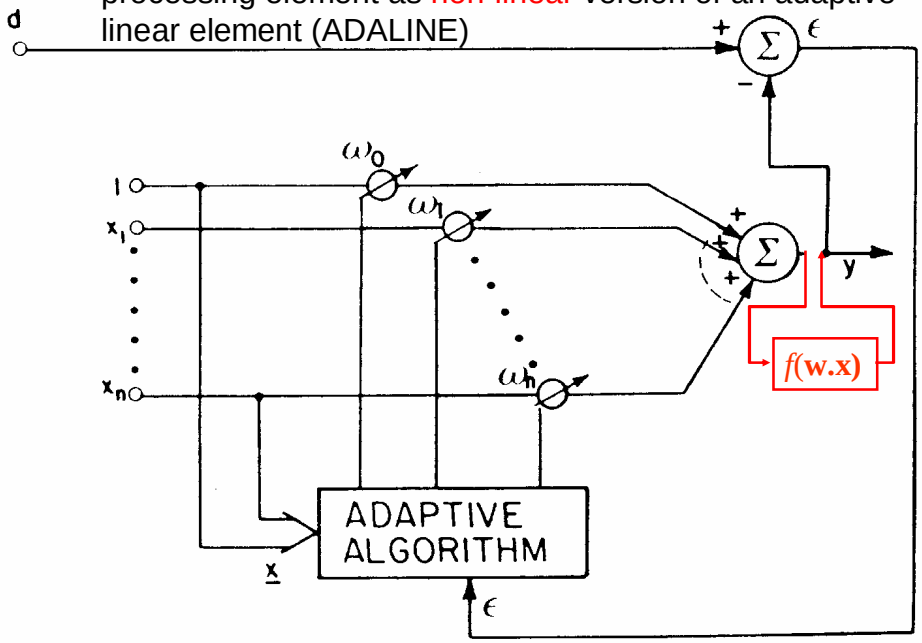


Some terms

- inputs of a processing element are described by an *input vector*, $\boldsymbol{x}$
- weights associated with input connections are described by a *weight vector*, $\boldsymbol{w}$
- *output signal*, y
- *transfer function*, f

for example: $y = f(\boldsymbol{w}, \boldsymbol{x}) = \sum_i w_i \cdot x_i = \boldsymbol{w} \cdot \boldsymbol{x}$

processing element as non-linear version of an adaptive linear element (ADALINE)



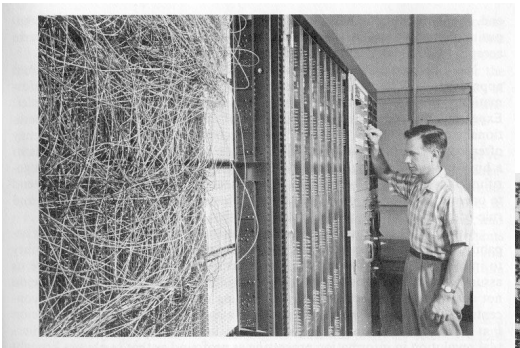


Fig. 1.6. • The Mark I Perceptron patchboard. The connection patterns were typically "random", so as to illustrate the ability of the perceptron to learn the desired pattern without need for precise wiring (in contrast to, unlike the precise wiring required in a programmed computer). Photo courtesy of Arvin Calspan Advanced Technology Center.

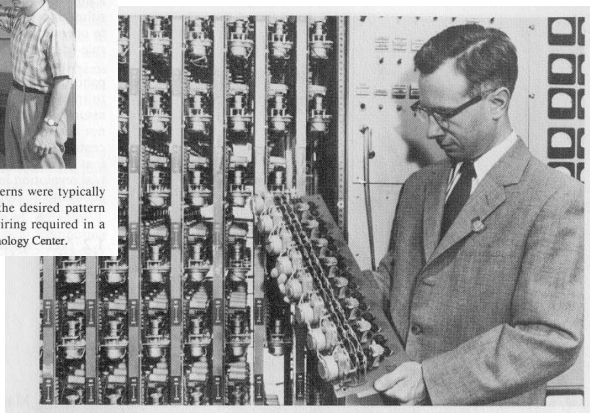
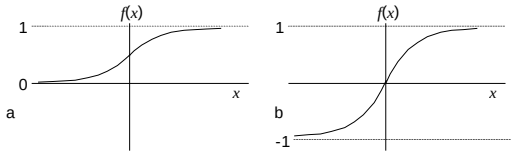


Fig. 1.5. • Charles Wightman holding a subrack of 8 motor/potentiometer pairs. Each motor/potentiometer pair functioned as a single adaptive weight value. The perceptron learning law was implemented in analog circuits that (when properly wired through the patchboard shown in Figure 1.6) would control the motor of each potentiometer (the resistance of which functioned to implement one weight). Photo courtesy of Arvin Calspan Advanced Technology Center.

often used transfer functions



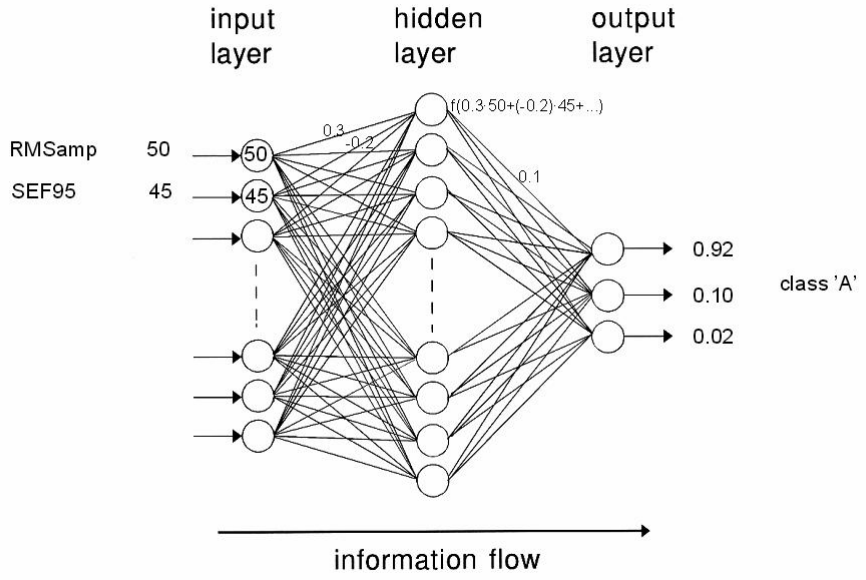
a sigmoid function

$$f(x) = \frac{1}{1+e^{-x}}$$

a hyperbolic tangent

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

An extra nonlinearity parameter, β, may be used to adjust the 'steepness' of the function, in which case the functions do not operate upon x, but on βx.

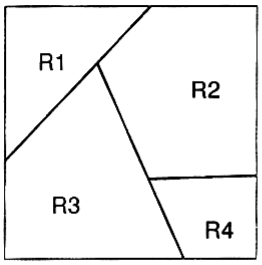


- often, one input is kept at a value of 1, the *bias*
- *local memory* contains *data variables* such as learning rate and momentum
- a *learning rule* influences the behaviour of the ANN by *adapting* the network *weights*

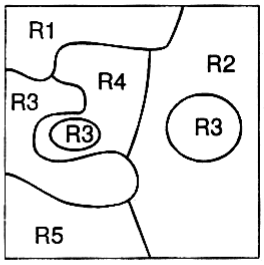
Use for Recognition and Classification

- generate suitable decision regions in feature space by looking for the right weight configuration

a: decision regions realised by a *linear* classifier
b: decision regions realised by a *non-linear* classifier



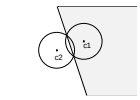
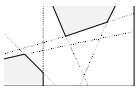
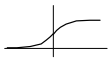
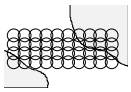
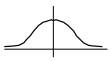
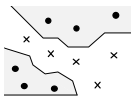
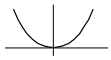
a



b

ANN-based classifiers

- Many artificial neural networks operate similarly to methods used in the statistical approach, i.e., they try to estimate decision regions in feature space. We can distinguish:
  - probabilistic classifiers that use *a priori* information about distributions and use supervised learning to estimate their parameters
  - hyperplane classifiers that construct hyperplane boundaries by calculating a weighted sum of inputs and passing this through a non-linear function
  - kernel (receptive-field) classifiers that establish receptive fields across feature space; if an input is near the centre of a receptive field of a processing unit, the output of that unit will be high
  - exemplar classifiers that use the distance between input pattern and training exemplars to come to classifications

group	estimation of decision regions	processing unit	examples of classifiers
probabilistic		distribution dependent	Gaussian mixture
hyperplane		sigmoid 	multi-layer perceptron networks
kernel		kernel 	radial basis networks
exemplar		(Euclid.)norm 	LVQ, ART, Kohonen networks

The first column gives the name of the group of ANN classifiers, the second column shows schematically for each group how the decision regions in feature space are estimated. The third column shows the kind of computations performed in the processing units and the fourth column contains examples of ANNs that belong to each group

Learning in ANNs

- Different groups of training/learning methods
  - supervised learning
  - unsupervised learning
  - reinforcement learning
- each group contains many methods

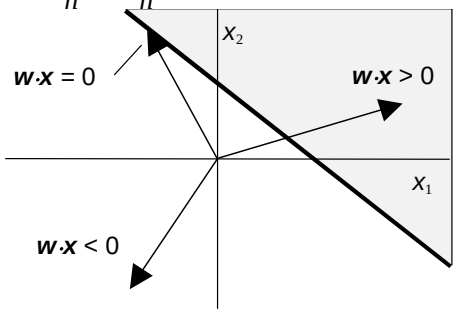
- examples discussed here:
  - *(generalised) delta rule* (supervised learning)
  - *Kohonen learning rule* (unsupervised learning)

Delta Rule
(aka Widrow, Widrow/Hoff, or LMS learning rule)

- originally used in the 60's in ADALINEs (ADAptive LINear Elements)
- in its generalised form very popular (backpropagation)

n-dimensional input vector ***x***

$$y = w_0 \cdot 1 + w_1 \cdot x_1 + \mathrm{L} + w_n \cdot x_n$$



hyperplane classifier

- input/desired output pair *k* is $(\boldsymbol{x}_k, y_k^*)$
- define cost function, *G*, as expectation of the squared error

$$G(\boldsymbol{w}) = E\left[(y_k^* - y_k)^2\right] \text{ or}$$

$$G(\boldsymbol{w}) = \lim_{N\to\infty}\left(\frac{1}{N}\right)\sum_{k=1}^{N}(y_k^* - y_k)^2$$

parabolic surface; 'slide' towards minimum by using $-\nabla_{\boldsymbol{w}} G$ (note: $\nabla_{\boldsymbol{w}}(-y_k) = -\boldsymbol{x}_k$)

$$\nabla_{\boldsymbol{w}} G(\boldsymbol{w}) = \lim_{N\to\infty}\left(\frac{1}{N}\right)\sum_{k=1}^{N} 2\cdot(y_k^* - y_k)\cdot(-\boldsymbol{x}_k)$$

$$\nabla_{\boldsymbol{w}} G(\boldsymbol{w}) = -2E\left[\delta_k \boldsymbol{x}_k\right]$$

δ_k is the error for input *k*

- this implies: average a large number of $\delta k\, \boldsymbol{x}k$ vectors, multiply by -2 and move weights in that direction
- Widrow & Hoff: update weights after every input presentation → **delta rule**:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha\,\delta_k\,\boldsymbol{x}_k$$

with learning rate α

alternatives:

- update only after a number of input presentations (**batched** version)
- use most recent weight update also in current weight update (**momentum** version):

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha\,\delta_k\,\boldsymbol{x}_k + \mu\,(\boldsymbol{w}_k - \boldsymbol{w}_{k-1})$$

with momentum μ

limited performance due to its **linearity**

ANN architectures

often designed to

- **store and retrieve information**
  associative networks (e.g., Hopfield networks)
  process ($\boldsymbol{x}_i$,$\boldsymbol{y}_i$*) examples →
  in response to $\boldsymbol{x}_i$ + **ε:** produce $\boldsymbol{y}_i$*
- **approximate mathematical functions**
  mapping networks (e.g., 'backpropagation' networks)
  process ($\boldsymbol{x}_i$,$\boldsymbol{y}_i$*) examples →
  approximate $f(\boldsymbol{x}) = \boldsymbol{y}^*$ for $\boldsymbol{x} \neq \boldsymbol{x}_i$
  map vectors from $\mathbf{R}^n$ to $\mathbf{R}^m$

Kolmogorov's theorem
any continuous function f:$[0,1]^n \rightarrow \mathbf{R}^m$, $f(\boldsymbol{x}) = \boldsymbol{y}$ can be approximated exactly by a three-layer ANN
(but how to create such an ANN???)

Multi-Layer Perceptron (MLP) trained with the backpropagation algorithm

- Perceptron: 'non-linear ADALINE'

$$y = f\left(\sum_{i=0}^{n} w_i \cdot x_i\right)$$

if the argument of f is larger than or equal to 0, f is +1, else it is -1
learning rule similar to delta rule

↓

decision regions built of 'half planes'

more complex decision regions by using multiple layers of perceptrons

Q: how to train such a configuration?
A: backpropagation

Backpropagation

- Minimise a cost function, *Gp*, defined for an input pattern *p* and a network with *m* outputs

$$G_p = \frac{1}{2}\sum_{j=1}^{m}(y_{pj}^{*} - y_{pj})^2$$

With y_{pj} the output of output element *j* when pattern *p* is presented. and y_{pj}* the desired output-
Again, we use the gradient descent to change the weights - the elements w_{ij} of weight vector $\mathbf{w}_j$ change according to:

$$\Delta_p w_{ij} \propto -\frac{\partial G_p}{\partial w_{ij}} = -\frac{\partial G_p}{\partial I_{pj}} \cdot \frac{\partial I_{pj}}{\partial w_{ij}}$$

I_{pj} is the input applied to processing element *j* as a result of presenting *p*

for output elements the weight change can be calculated straight-forwardly using

$$I_{pj} = \sum_{i=0}^{k} w_{ij} \cdot y_{pi}$$

We would like to use the following weight update rule:

$$\Delta_p w_{ij} = \alpha \cdot \delta_{pj} \cdot y_{pi}$$

this is called the **generalised delta rule**

with learning rate α (usually decreasing with time), and the error made by processing element *j* as a result of the application of pattern *p* denoted as δ_{pj}

For output elements, we can easily calculate δ_{pj} since we know what their desired and actual output (and thus, error) is.

$$\Delta_p w_{ij} = -\frac{\partial G_p}{\partial I_{pj}} \frac{\partial\left(\sum_i w_{ij} \cdot y_{pi}\right)}{\partial w_{ij}} = -\frac{\partial G_p}{\partial I_{pj}} \cdot y_{pi}$$

thus, if we would like to have a rule of the form $\Delta_p w_{ij} = \alpha \cdot \delta_{pj} \cdot y_{pi}$, we need to estimate δ_{pj} as:

$$\delta_{pj} \propto -\frac{\partial G_p}{\partial I_{pj}} = -\frac{\frac{1}{2}\partial\left(\sum_j (y^*_{pj} - y_{pj})^2\right)}{\partial I_{pj}}$$

$$= -\frac{1}{2} \cdot 2 \cdot (y^*_{pj} - y_{pj}) \cdot -\frac{\partial y_{pj}}{\partial I_{pj}}$$

$$= (y^*_{pj} - y_{pj}) \cdot f_j'(I_{pj})$$

Eventually we get for the output elements:

$$\delta_{pj} = (y^*_{pj} - y_{pj}) \cdot f_j'(I_{pj})$$

For 'hidden' nodes however, this is a bit more complex - we have to apply the chain rule for differentiation and use the error of the output elements. We thus 'backpropagate' the error through the network to calculate all weight updates.
For the elements in the hidden layer(s) we get

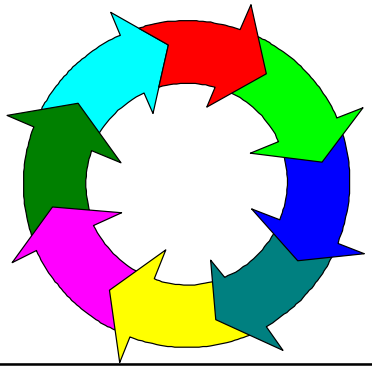
$$\delta_{pj} = \sum_{k=1}^{m} \delta_{pk} \cdot w_{jk} \cdot f_j'(I_{pj})$$

f'_j is the derivative of the transfer function of processing element j
Note: since the derivative of the transfer function is used, this function must be differentiable for every input value.

Backpropagation training

- with the weight update rule and the expressions for δ_{pj} we have the tools to calculate weight changes for every processing element after each pattern presentation *p*.

- Iterative training process



•present a new input pattern

•calculate network output using summations and transfer functions

•calculate errors of output elements

•calculate errors of hidden elements (backpropagation)

•update weights using generalised delta rule

Backpropagation process

- this iterative process will eventually lead us to a weight configuration that is associated with the global minimum in the, very-many-dimensional, cost-function surface (or, at least we hope so)
- There is no guarantee that we will actually reach the global minimum - the process can get stuck in local minima as well
- The training process can be very time-consuming - often training is stopped when a predefined number of iterations has been reached, the error on the training set or the 'training evaluation set 'drops below a certain threshold, or the weights do no significantly change over a long time.
- Again, variations like 'batched weight updating 'and use of momentum terms may be used to speed up the process.

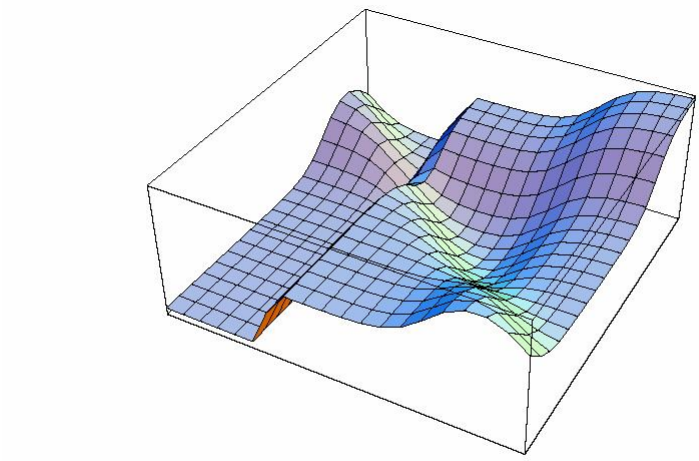
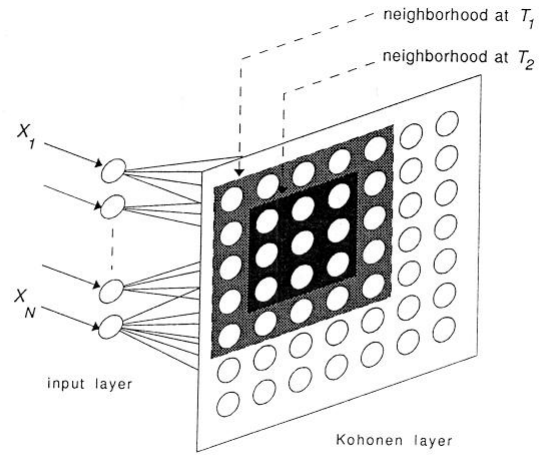


Fig. 7.5. A local minimum of the error function

Kohonen learning rule

- unsupervised, i.e., no desired outputs
- allows creation of an estimation of the probability density function (PDF) of applied inputs
- two layers;
  - input layer (with *N* elements)
  - output (Kohonen) layer (*M* elements)

Kohonen network



output y_i of Kohonen element i:

$$y_i = D(\boldsymbol{w}_i, \boldsymbol{x})$$

depends on distance between $\boldsymbol{w}_i$ and $\boldsymbol{x}$

'competition' $\rightarrow$ one 'winner':
element with smallest distance: $yc = 1$
all others: $yj = 0$

kohonen learning rule:

$$\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) + h_{cj}(t) \cdot (\boldsymbol{x}_j(t) - \boldsymbol{w}_j(t))$$

with neighborhood kernel h_{cj}

$$h_{cj}(t) = h(\left\| \boldsymbol{r}_c - \boldsymbol{r}_j \right\|, t)$$

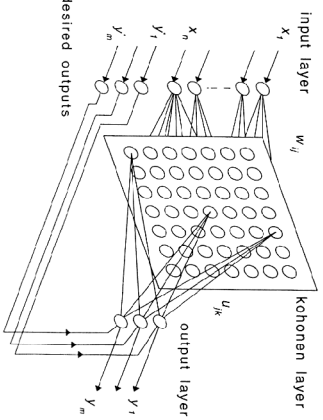
with $\boldsymbol{r_c}$ the location of the winner in the Kohonen layer
often:

$$h_{cj}(t) = \alpha(t)$$

with $\alpha(t) > 0$ for nodes in the 'neighborhood' of c, and 0 for all others
both the neighborhood size and α decrease as function of time

Counterpropagation Network

(Kohonen network extended with an extra output layer)



processing element k in the extra output layer is fully connected to all M Kohonen elements:

$$y_{pk} = \sum_{j=1}^{M} u_{jk} \cdot z_{pj}$$

z_{pj} is the output of Kohonen element j in reaction to pattern p, it can be 0 or 1. ujk is the weight between Kohonen element j and output element k

its weights are updated using:

$$\Delta_p u_{jk} = \alpha \cdot (y^*_{pk} - y_{pk}) \cdot z_{pj}$$

only weights connected to the winning Kohonen element are updated.

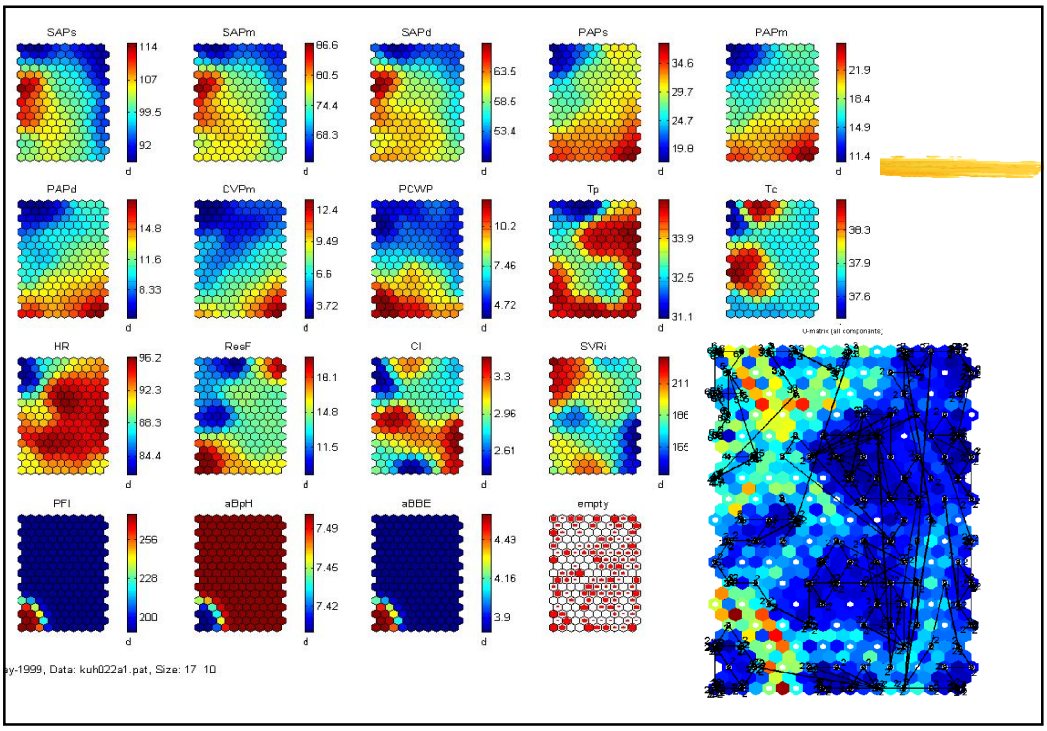
Each weight will eventually converge to the average desired output associated with inputs that cause a certain Kohonen element to win the competition

Example: following patient state in ICU

- z from the routinely monitored data;
  - y SAPs,m,d PAPs,m,d
  - y CVPm, PCWP
  - y Tp, Tc
  - y HR ResF CI SVRi
  - y SpO2 PFI aBpH aBBE

18-dim vectors measured at 2 min.intervals

each vector labelled with the most recent Ramsay score known at that moment



Other types of networks

- The backpropagation network and the Kohonen and counterpropagation networks are only a few examples.
- A wide range of ANN paradigms exists, some examples:
  - stochastic networks (e.g., try to get out of local minima using a technique called simulated annealing
  - spatiotemporal networks (include time aspects, e.g., for speech recognition)
  - multi-modular networks (divide a huge problem into simpler subproblems and solve each of those with an ‘expert’ network)
  - combinations with other techniques, such as expert systems or fuzzy logic (‘hybrid networks’)

Usage

- In non-trivial applications ANNs are typically used as a module co-operating with other modules that may use other techniques (rules, algorithms, …) in so-called hybrid systems. Rarely an ANN can be used to solve ‘everything’

Sufficient usable data is crucial

Large data sets are needed for training note:

- A data set with 999.998 occurrences of disease A and only 2 occurrences of disease B may be a small data set (quite often we would like the ANN to help us exactly with identifying those rare disease B cases)
- Missing data often forms a serious problem in clinical applications