Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks

Secure communication based on noisy input data

Feature extraction and context computing

Stephan Sigg

May 3, 2011

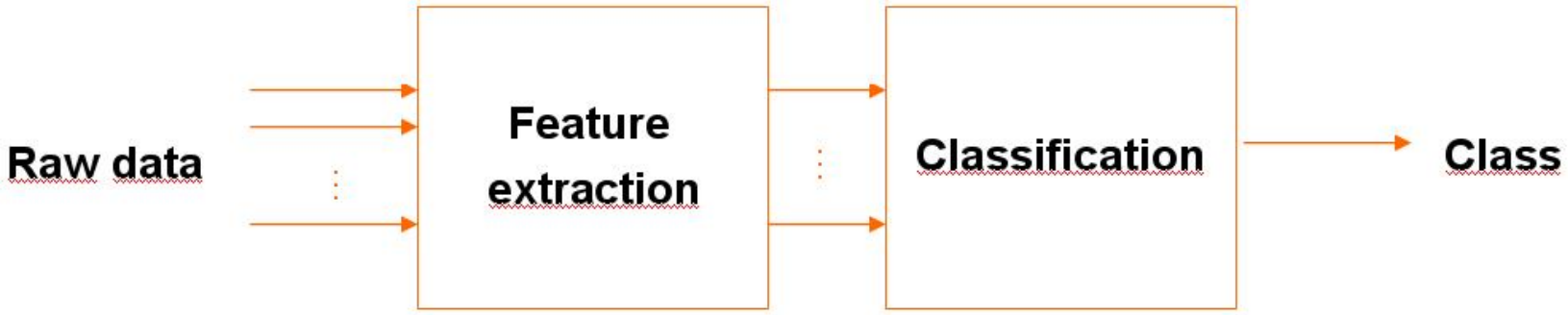
Overview and Structure

- Classification methods
- Feature extraction
  - Features from audio
  - Features from RF
- Fuzzy Commitment
- Fuzzy Extractors
- Authentication with noisy data
- Error correcting codes
- Entropy
- Physically unclonable functions

Outline

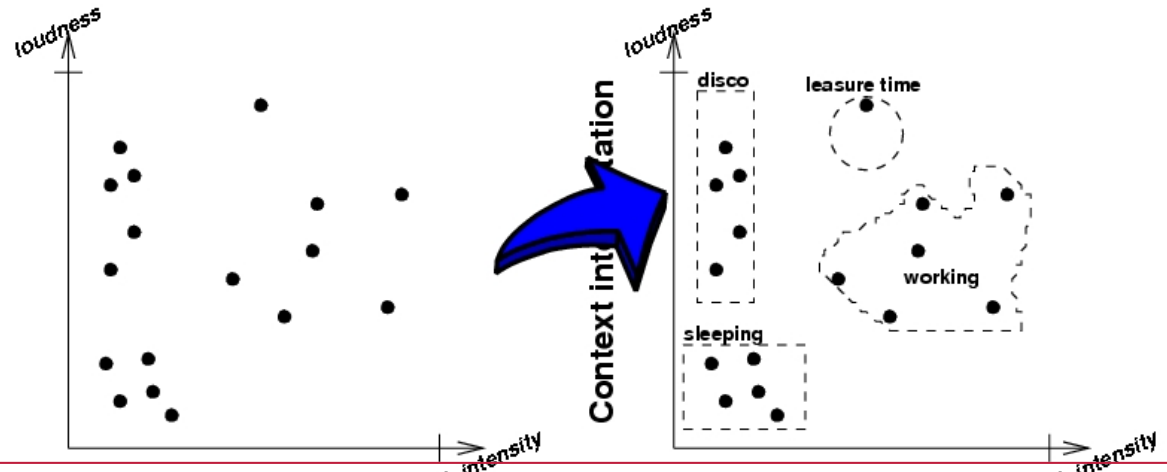
Pattern recognition and classification

- Classical pattern recognition
  - Mapping of features onto classes by utilisation of prior knowledge
  - What are characteristic features?
  - Which approaches are suitable to obtain these features?



Pattern recognition and classification

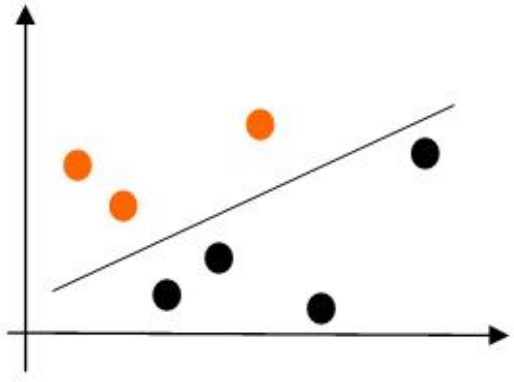
- From features to context
  - Measure available data on features
  - Context reasoning by appropriate method
    - Syntactical (rule based – e.g. RuleML)
    - Bayesian classifier
    - Non-parametric
    - Linear discriminant
    - Neural networks
    - Sequential
    - Stochastic



Pattern recognition and classification

- Allocation of sensor value by defined function
  - Correlation of various data sources
  - Several methods possible – simple approaches
  - Template matching
  - Minimum distance methods
  - 'Integrated' feature extraction
    - Nearest Neighbour
    - Neural Networks
- Problem
  - Measured raw data might not allow to derive all features required
  - Therefore often combination of sensors

Not enough features



Sufficient feature count

Pattern recognition and classification

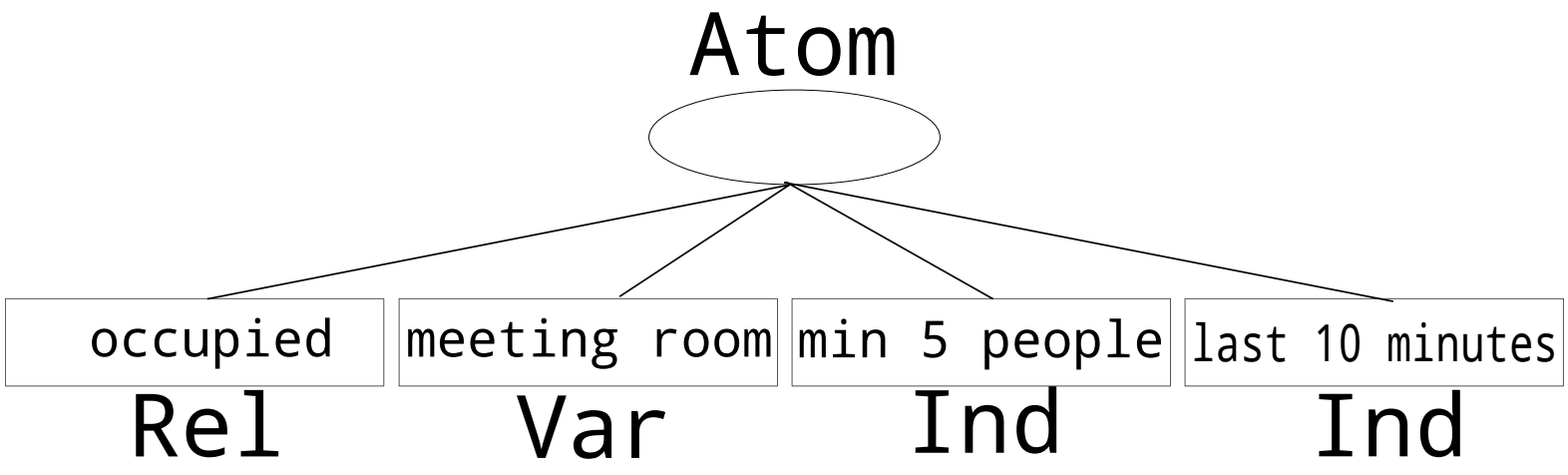
- Methods – Syntactical (Rule based)
  - Idea: Description of Situation by formal Symbols and Rules
  - Description of a (agreed on?) world view
  - Example: RuleML
- Comment
  - Pro:
    - Combination of rules and identification of loops and impossible conditions feasible

    Contra:
    - Very complex with more elaborate situations
    - Extension or merge of rule sets typically not possible without contradictions

Pattern recognition and classification

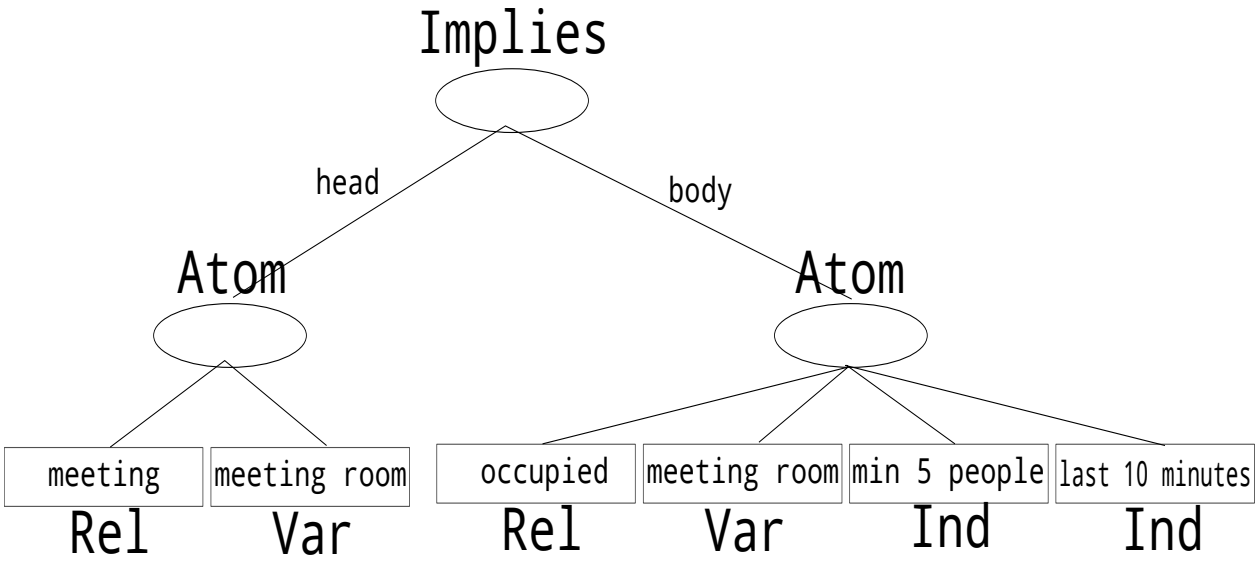
- Rule Markup Language: Language for publishing and sharing rules
- Hierarchy of rule-sub-languages (XML, RDF, XSLT, OWL)
- Example:
  - A meeting room was occupied by min 5 people for the last 10 minutes.

```
<Atom>
     <Rel> occupied </Rel>
     <Var> meeting room </Var>
     <Ind> min 5 persons </Ind>
     <Ind> last 10 minutes </Ind>
</Atom>
```



Pattern recognition and classification

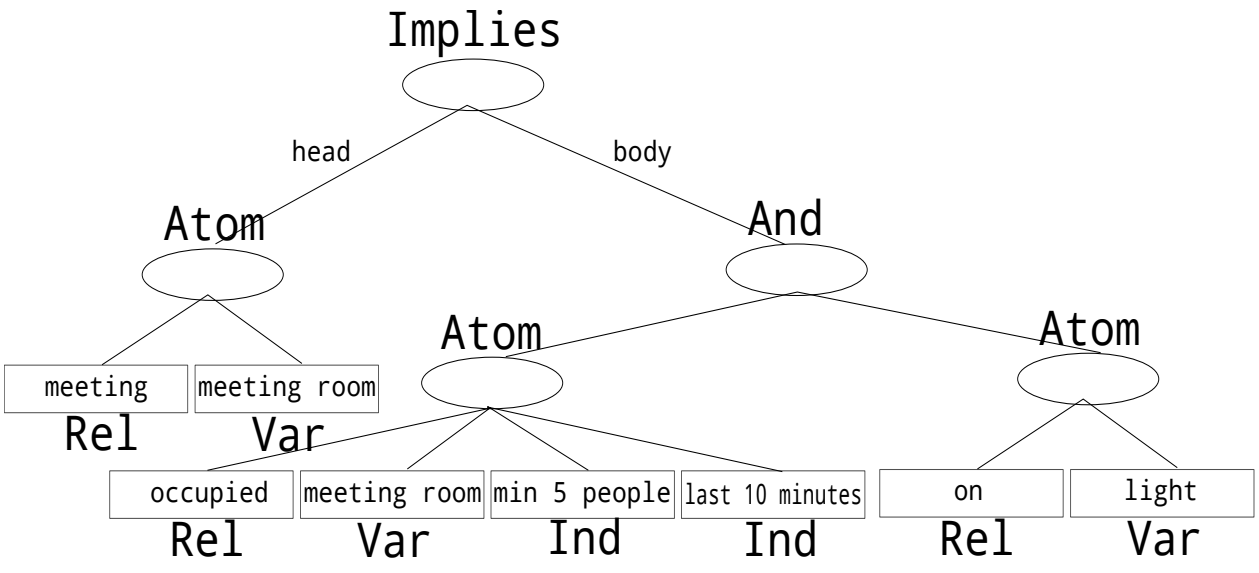
- Also conditions can be modelled
  - A Meeting is taking place in a meeting room when it was occupied by min 5 people for the last 10 minutes.




```
<Implies>
  <head>
     <Atom>
          <Rel> meeting </Rel>
          <Var> meeting room </Var>
     </Atom>
  </head>
  <body>
     <Atom>
          <Rel> occupied </Rel>
          <Var> meeting room </Var>
          <Ind> min 5 persons </Ind>
          <Ind> last 10 minutes </Ind>
     </Atom>
  </body>
</Implies>
```

Pattern recognition and classification

- Logical combination of conditions
  - A Meeting is taking place in a meeting room when it was occupied by min 5 people for the last 10 minutes and the light is on.

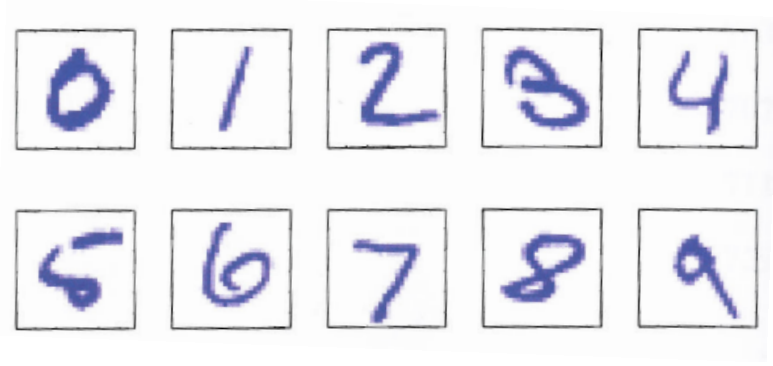



```
<Implies>
   <head>
      <Atom>
         <Rel> meeting </Rel>
         <Var> meeting room </Var>
      </Atom>
   </head>
   <body>
      <And>
         <Atom>
            <Rel> on </Rel>
            <Var> light </Var>
         </Atom>
         <Atom>
            <Rel> occupied </Rel>
            <Var> meeting room </Var>
            <Ind> min 5 persons </Ind>
            <Ind> last 10 minutes </Ind>
         </Atom>
      </And>
   </body>
</Implies>
```

Outline

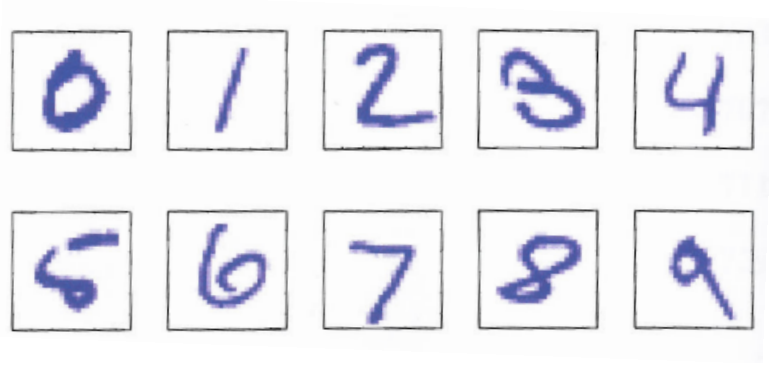
Recognition of patterns

- Patterns can be recognised by stating a sufficient number of rules
- Sensor readings are inaccurate
- Very high count of rules to accurately model all variations of sensor readings that belong to one class
- Therefore: Consider machine learning approaches



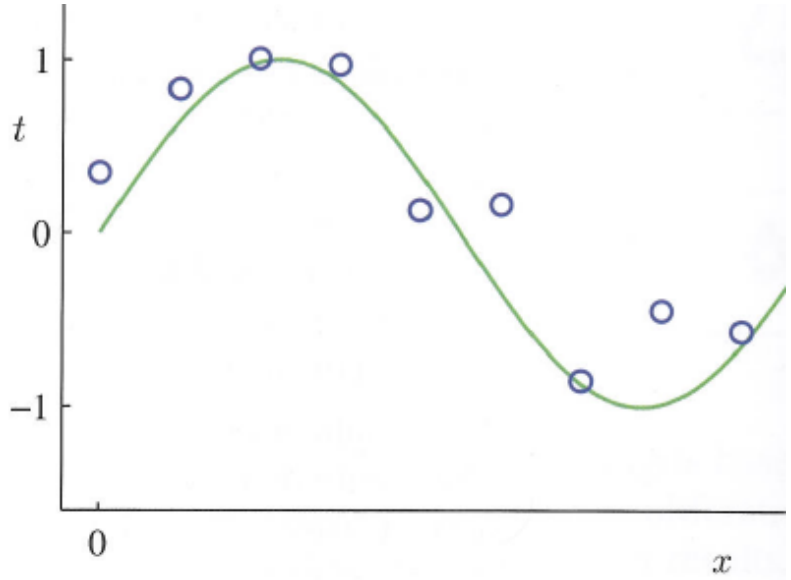
Recognition of patterns

- A training set $x_1 \ldots x_N$ of a large number of N samples is utilised
- Classes $t_1 \ldots t_N$ of all samples in this set known in advance
- A machine learning algorithm computes a function $y(x)$ and generates a new target t‘



Polynomial curve fitting

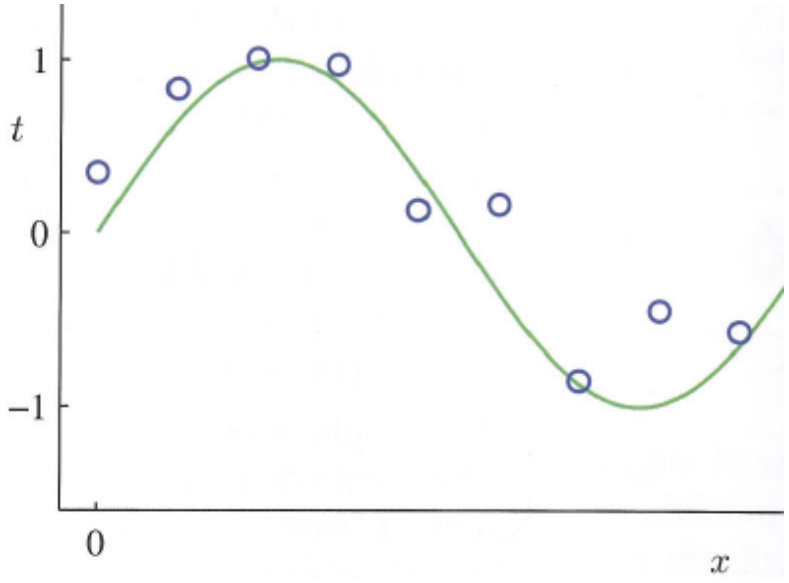
- As an example we assume that a curve shall be approximated by a machine learning approach
- Sample points are created for the function $\sin(2\pi x) + \mathcal{N}$ where $\mathcal{N}$ is a random noise value



Polynomial curve fitting

- We will try to fit the data points into a polynomial function of the form

$$y(x, \vec{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$



Polynomial curve fitting

- We will try to fit the data points into a polynomial function of the form

$$y(x, \vec{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

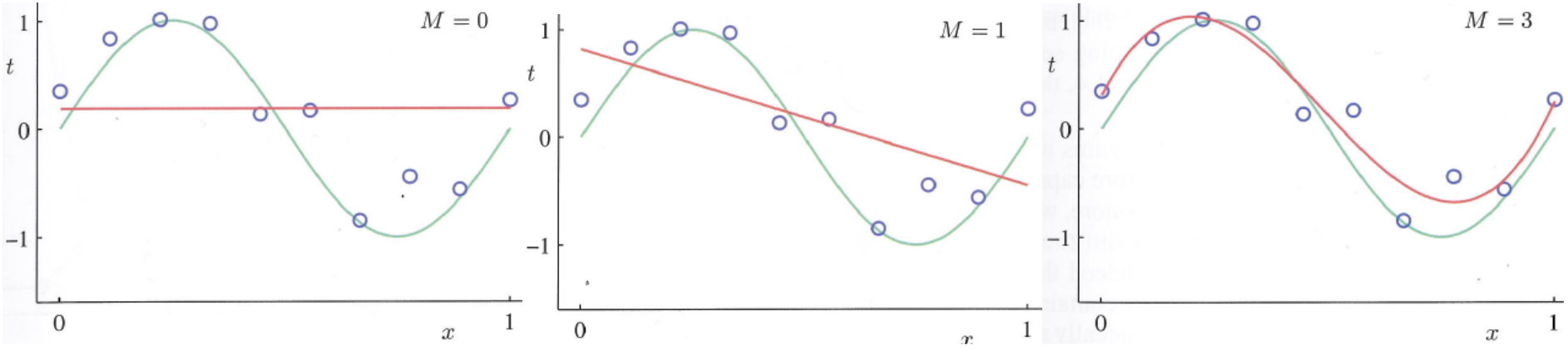
- This can be obtained by minimising an error function that measures the misfit between $y(x, \vec{w})$ and the training data set:

$$E(\vec{w}) = \frac{1}{2} \sum_{i=1}^{N} \left[y(x_i, \vec{w}) - t_i \right]^2$$

- $E(\vec{(}w))$ is non-negative and zero if and only if all points are covered by the function

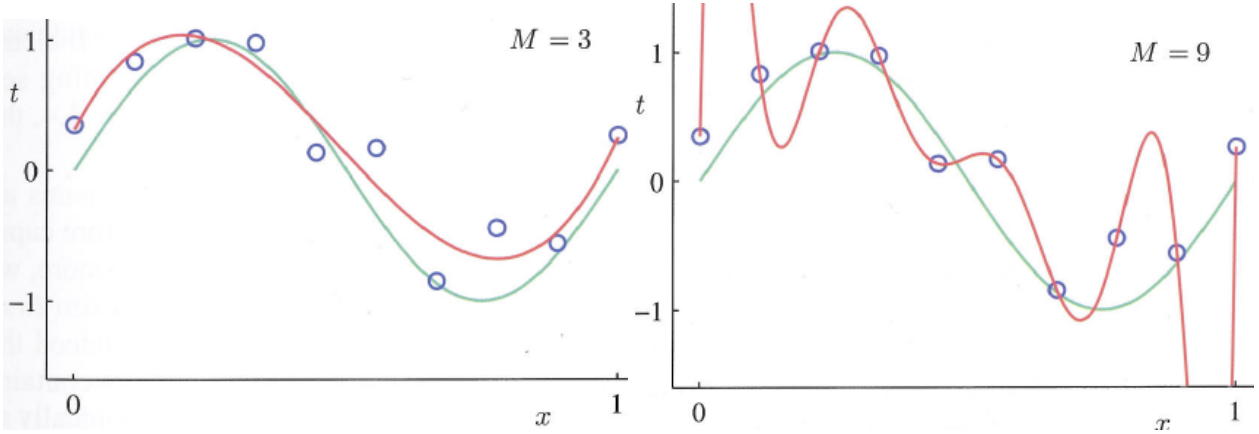
Polynomial curve fitting

- One problem is the right choice of the dimension M of the polynomial
- When M is too small, the accuracy of the approximation might be bad



Polynomial curve fitting

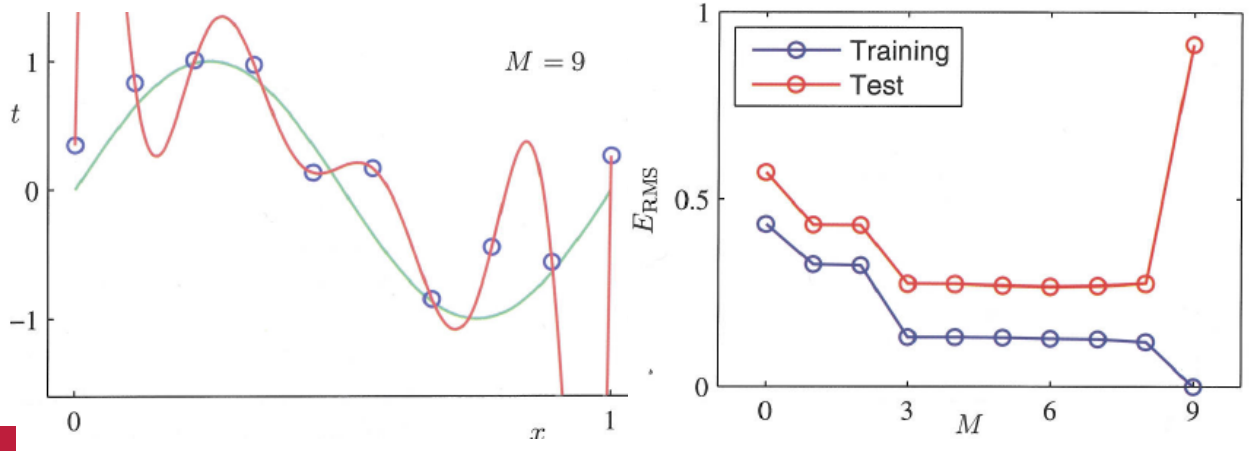
- One problem is the right choice of the dimension M of the polynomial
- In particular, when M becomes too big, the resulting polynomial will cross all points exactly
- When M reaches the count of samples in the training data set, it is always possible to create a polynomial of order M that contains all values in the data set exactly.



Polynomial curve fitting

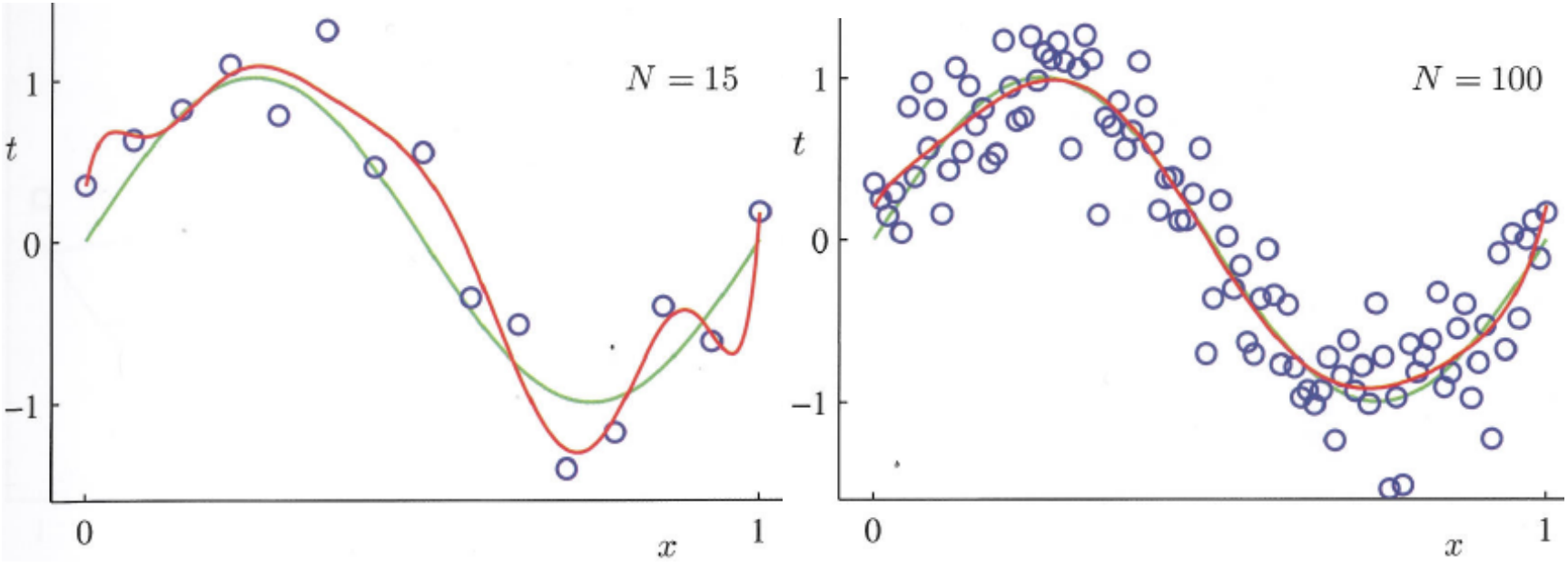
- This event is called overfitting
- The polynomial now trained too well to the training data
- It will therefore perform badly on another sample of test data for the same phenomenon
- We can visualise this by the Root of the Mean Square (RMS) of $E(\vec{w})$

$$E_{RMS} = \sqrt{\frac{2E(\vec{w})}{N}}$$



Polynomial curve fitting

- With increasing number of data points, the problem of overfitting becomes less severe for a given value of M



Polynomial curve fitting

- One solution to cope with overfitting is regularisation
- A penalty term is added to the error function
- This term discourages the coefficients of $\vec{w}$ from reaching large values

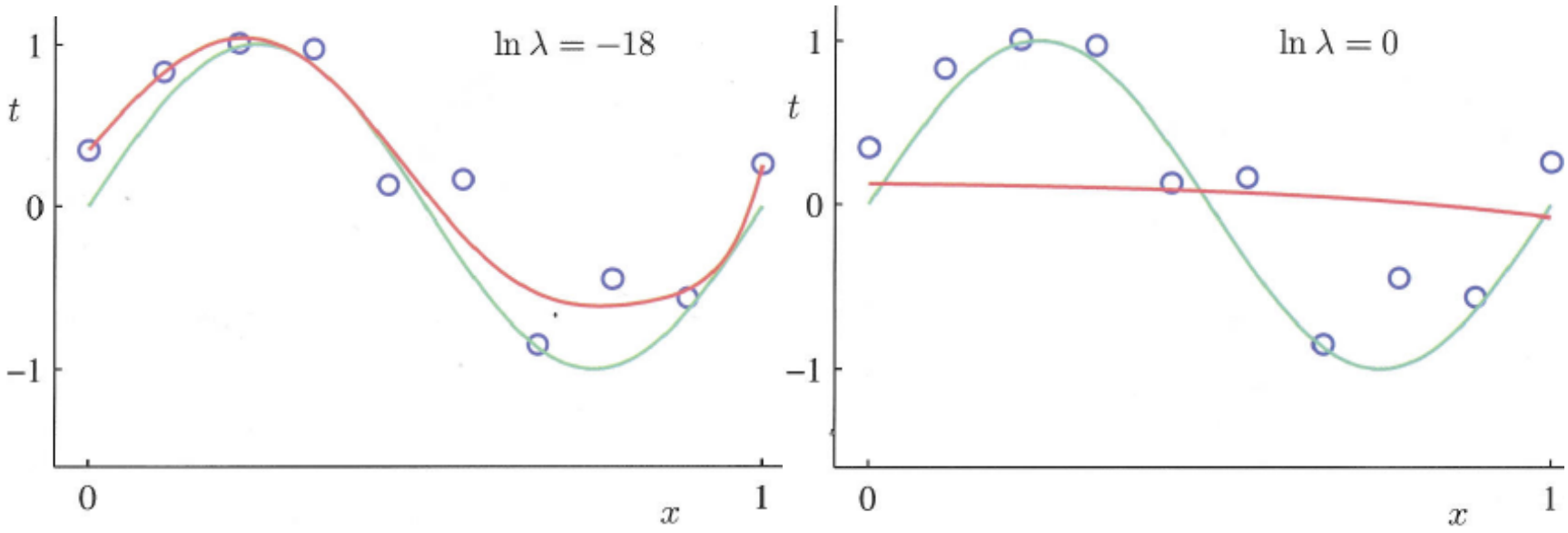
$$\overline{E}(\vec{w}) = \frac{1}{2}\sum_{i=1}^{N}\left[y(x_i, \vec{w}) - t_i\right]^2 + \frac{\lambda}{2}||\vec{w}||^2$$

with

$$||\vec{w}||^2 = \vec{w}^T\vec{w} = w_0^2 + w_1^2 + \cdots + w_M^2$$

Polynomial curve fitting

- Depending on the value of λ, overfitting is controlled



$$\overline{E}(\vec{w}) = \frac{1}{2}\sum_{i=1}^{N}\left[y(x_i, \vec{w}) - t_i\right]^2 + \frac{\lambda}{2}||\vec{w}||^2$$

Outline

Bayesian decision theory

- With probability theory, the probability of events can be estimated by repeatedly generating events and counting their occurrences
- When, however, an event only very seldom occurs or is hard to generate, other methods are required
- Example: Probability that the Arctic ice cap will have disappeared by the end of this century
- In such cases, we have to find a way to model uncertainty.
- In fact, it is possible to represent uncertainty by probability

Conditional probability

Conditional probability

The conditional probability of two events χ_1 and χ_2 with $P(\chi_2) > 0$ is denoted by $P(\chi_1|\chi_2)$ and is calculated by

$$P(\chi_1|\chi_2) = \frac{P(\chi_1 \cap \chi_2)}{P(\chi_2)}$$

$P(\chi_1|\chi_2)$ describes the probability that event χ_2 occurs in the presence of event χ_2.

Bayesian decision theory

- With the notion of conditional probability we can express the effect of observed data $\overrightarrow{t} = t_1, \ldots, t_N$ on a probability distribution of $\overrightarrow{w}$: $P(\overrightarrow{w})$:

$$P(\overrightarrow{w}|\overrightarrow{t}) = \frac{P(\overrightarrow{t}|\overrightarrow{w})P(\overrightarrow{w})}{P(\overrightarrow{t})}$$

- Thomas Bayes described a way to evaluate the uncertainty of $\overrightarrow{w}$ after observing $\overrightarrow{t}$
- $P(\overrightarrow{t}|\overrightarrow{w})$ expresses how probable a value for $\overrightarrow{t}$ is given a fixed choice of $\overrightarrow{w}$

Bayesian decision theory

- A principle difference between the Bayesian viewpoint and a frequentist viewpoint is that prior assumptions are provided for a model
- Example:
  - Consider a fair-looking coin that scores heads in three consecutive coin tosses.
  - A classical maximum likelihood estimate will result in predict head for future coin tosses with probability 1
  - A Bayesian approach will include prior assumptions on the probability of events and would result in a less extreme conclusion

Bayesian curve fitting

- In the curve fitting problem, we are given $\vec{x}$ and $\vec{t}$ together with a new sample x_{M+1}
- The task is to find a good estimation of the value t_{M+1}
- This means that we want to evaluate the predictive distribution

$$p(t_{M+1}|x_{M+1}, \vec{x}, \vec{t})$$

- To account for measurement inaccuracies, typically also a probability distribution (e.g. Gauss) is underlying the sample vector $\vec{x}$

Bayesian curve fitting

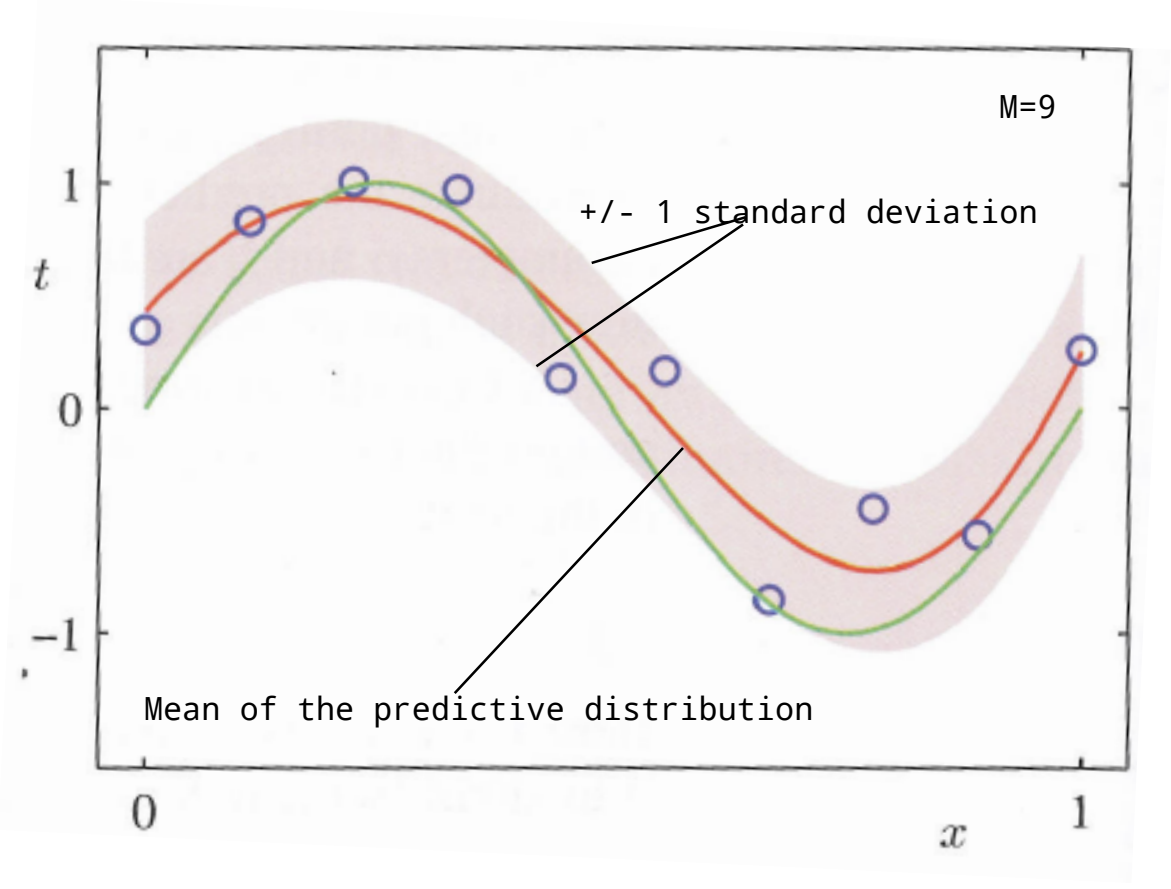
- This means that we want to evaluate the predictive distribution

$$p(t_{M+1}|x_{M+1}, \overrightarrow{x}, \overrightarrow{t})$$

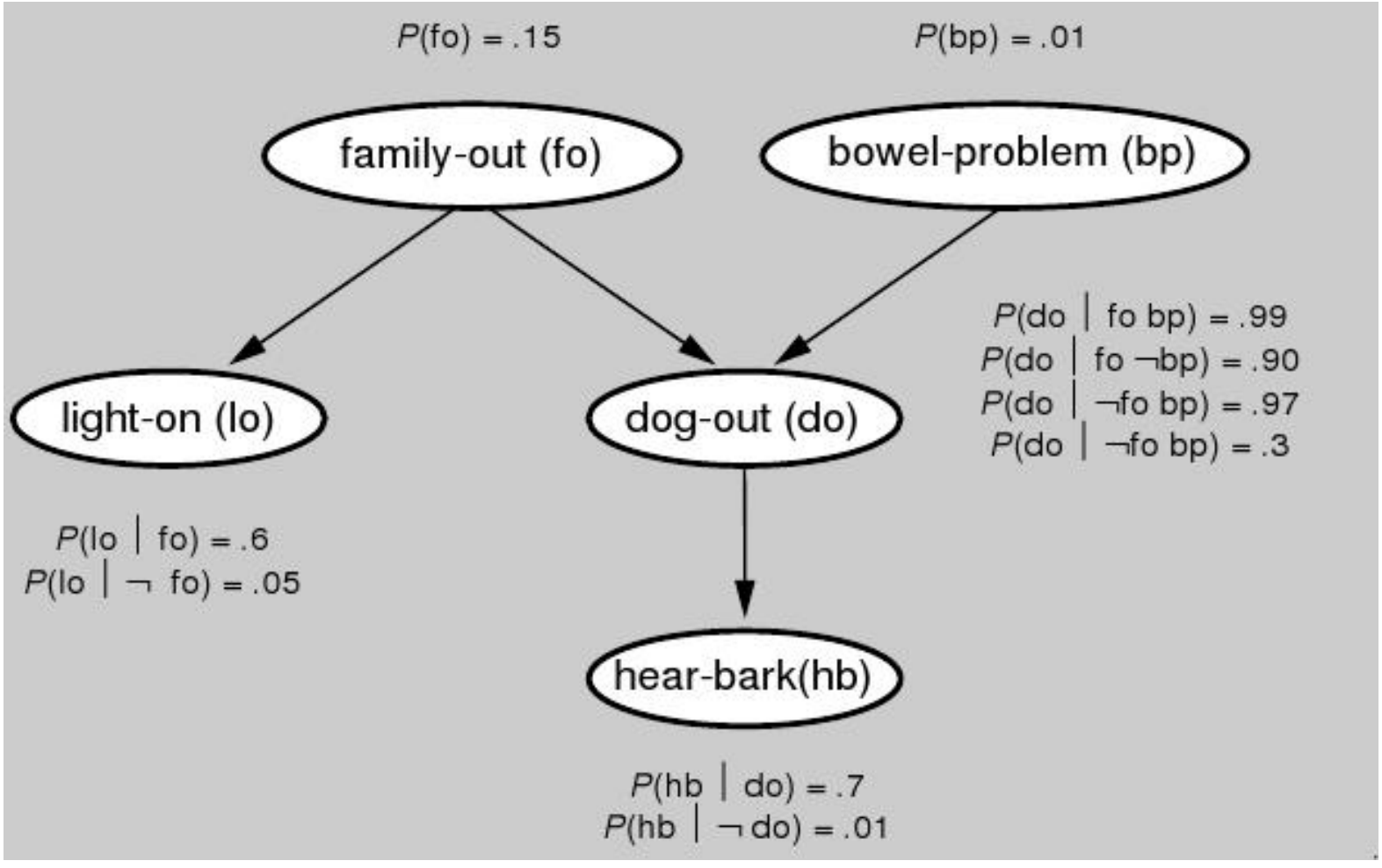
- After consistent application of the sum and product rules of probability we can rewrite this as

$$p(t_{M+1}|x_{M+1}, \overrightarrow{x}, \overrightarrow{t}) = \int p(t_{M+1}|x_{M+1}, \overrightarrow{w}) p(\overrightarrow{w}|\overrightarrow{x}, \overrightarrow{t}) d\overrightarrow{w}$$

Bayesian curve fitting

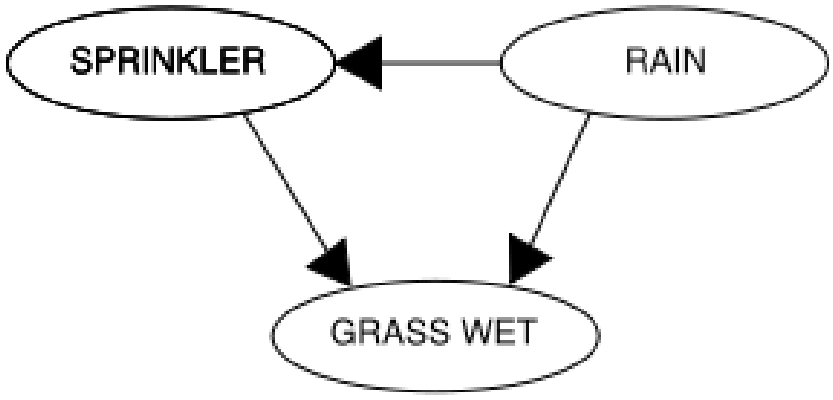


Example


P(fo) = .15
P(bp) = .01
family-out (fo)
bowel-problem (bp)
light-on (lo)
dog-out (do)
hear-bark(hb)
P(do | fo bp) = .99
P(do | fo ¬bp) = .90
P(do | ¬fo bp) = .97
P(do | ¬fo bp) = .3
P(lo | fo) = .6
P(lo | ¬ fo) = .05
P(hb | do) = .7
P(hb | ¬ do) = .01

Example

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



	RAIN	
T	F	
0.2	0.8	

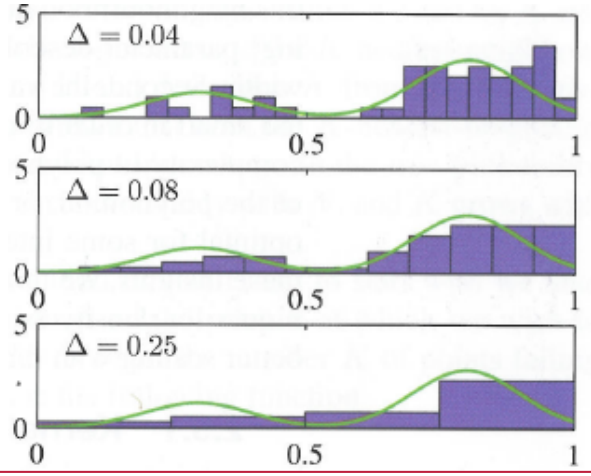
		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Outline

Histogram methods

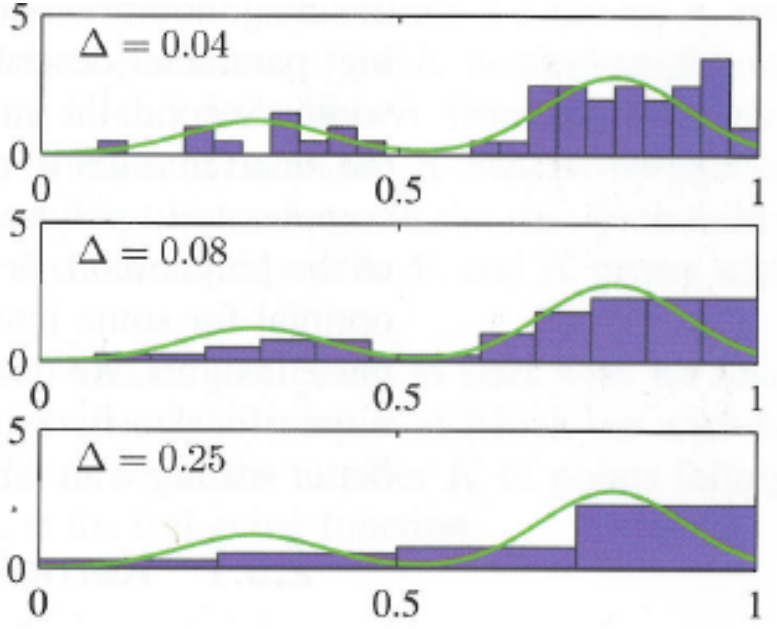
- An alternative approach to function estimation are histogram methods
- In general, the probability density of an event is estimated by dividing the range of N values into bins of size Δ_i
- Then, count the number of observations that fall inside bin Δ_i
- This is expressed as a normalised probability density

$$p_i = \frac{n_i}{N\Delta_i}$$



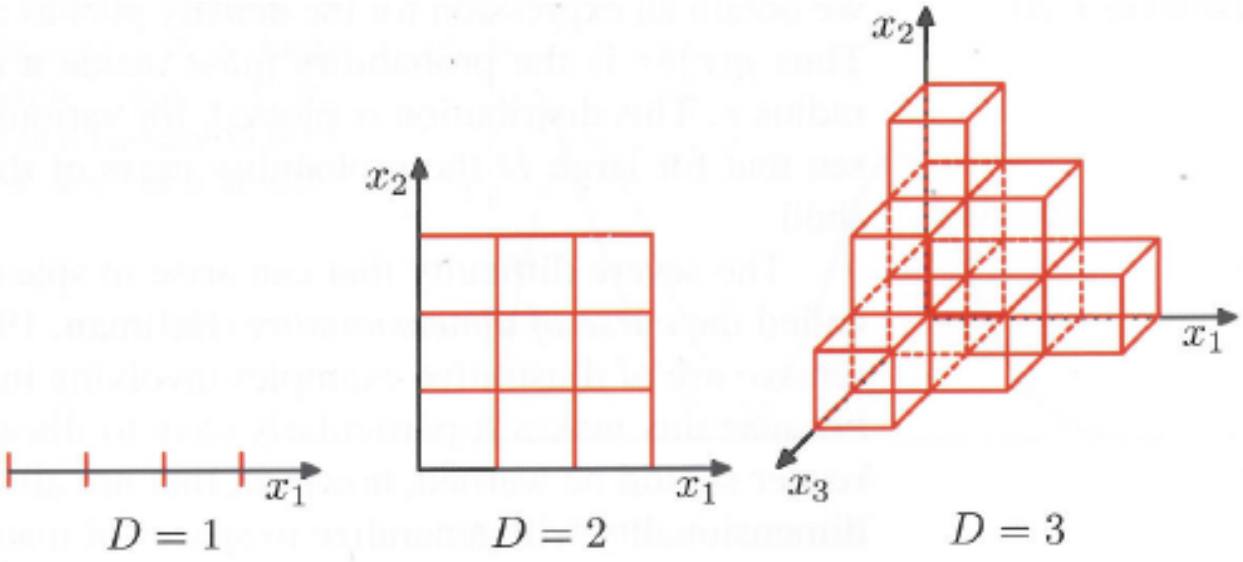
Histogram methods

- The accuracy of the estimation is dependent on the width of the bins
- The approach is well suited also for huge amounts of data since the data items can be discarded once the histogram is created



Histogram methods

- Issues:
  - Due to the edges of the bins, the modelled distribution is characterised by discontinuities not present in the underlying distribution observed
  - The method does not scale well with increasing dimension (Curse of dimensionality)



Parzen estimator methods

- Assume an unknown probability density $p(\cdot)$
- We want to estimate the probability density $p(\overrightarrow{x})$ of $\overrightarrow{x}$ in a $\mathcal{D}$-dimensional Euclidean space
- We consider a small region $\mathcal{R}$ around $\overrightarrow{x}$:

$$P = \int_{\mathcal{R}} p(\overrightarrow{x}) d\overrightarrow{x}$$

Parzen estimator methods

- We utilise a data set of N observations
- Each observation has a probability of P to fall inside $\mathcal{R}$
- With the binomial distribution we can calculate the count K of points falling into $\mathcal{R}$:

$$\mathrm{Bin}(K|N,P) = \frac{N!}{K!(N-K)!}P^K(1-P)^{N-K}$$

Parzen estimator methods

- We utilise a data set of N observations
- Each observation has a probability of P to fall inside $\mathcal{R}$
- With the binomial distribution we can calculate the count K of points falling into $\mathcal{R}$:

$$\mathrm{Bin}(K|N,P) = \frac{N!}{K!(N-K)!}P^K(1-P)^{N-K}$$

- For large N we can show

$$K \approx NP$$

- With sufficiently small $\mathcal{R}$ we can also show for the volume V of $\mathcal{R}$

$$P \approx p(\vec{x})V$$

- Therefore, we can estimate the density as

$$p(\vec{x}) = \frac{K}{NV}$$

Parzen estimator methods

- We assume that $\mathcal{R}$ is a small hypercube
- In order to count the number K of points that fall inside $\mathcal{R}$ we define

$$k(\overrightarrow{u}) = \begin{cases} 1, & |u_i| \leq \frac{1}{2}, \quad i = 1, \ldots, D, \\ 0, & \text{otherwise} \end{cases}$$

- This represents a unit cube centred around the origin
- This function is an example of a kernel-function or Parzen window

Parzen estimator methods

$$k(\vec{u}) = \begin{cases} 1, & |u_i| \leq \frac{1}{2}, \quad i = 1, \ldots, D, \\ 0, & \text{otherwise} \end{cases}$$

- When the measured data point $\vec{x_n}$ lies inside a cube of side h centred around $\vec{x}$, we have

$$k\left(\frac{\vec{x} - \vec{x_n}}{h}\right) = 1$$

- The total count K of points that fall inside this cube is

$$K = \sum_{n=1}^{N} k\left(\frac{\vec{x} - \vec{x_n}}{h}\right)$$

Parzen estimator methods

- The total count K of points that fall inside this cube is

$$K = \sum_{n=1}^{N} k\left(\frac{\vec{x} - \vec{x_n}}{h}\right)$$

- When we substitute this in the density estimate derived above

$$p(\vec{x}) = \frac{K}{NV}$$

- with volume $V = h^D$ we obtain the overall density estimate as

$$p(\vec{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} \left(\frac{\vec{x} - \vec{x_n}}{h}\right)$$

Parzen estimator methods

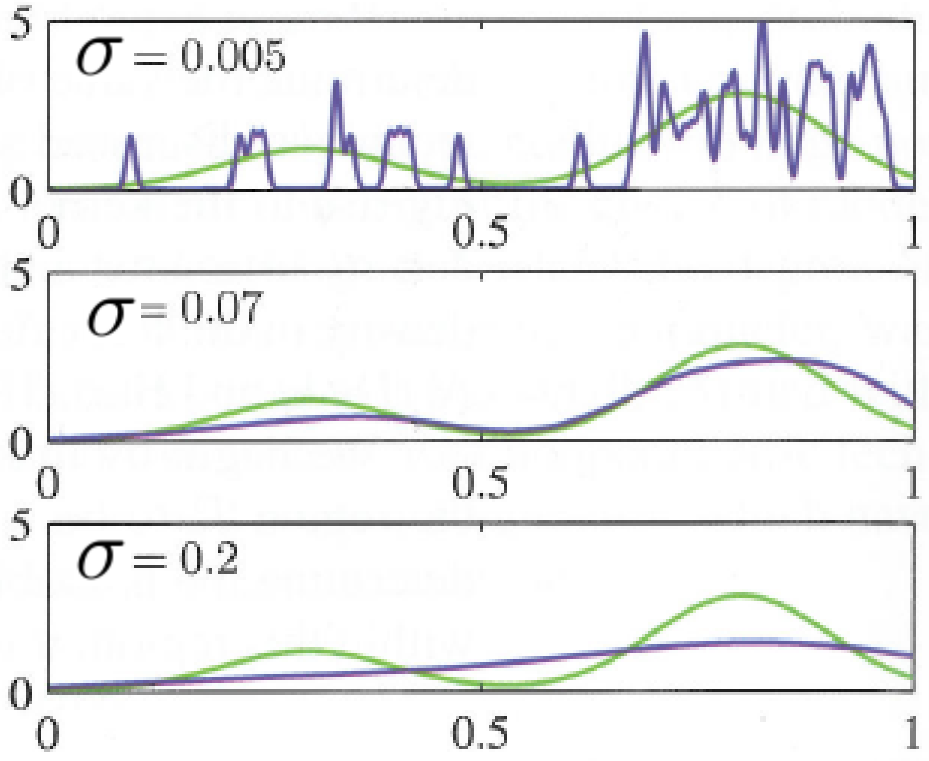
$$p(\vec{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} \left(\frac{\vec{x} - \vec{x_n}}{h} \right)$$

- Again, this density estimator suffers from artificial discontinuities
  - Due to the fixed boundaries of the cubes
- This problem can be overcome by choosing a smoother kernel function
  - A common choice is a Gaussian kernel with a standard deviation σ

$$p(\vec{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{\frac{D}{2}}} e^{-\frac{||\vec{x} - \vec{x_n}||^2}{2\sigma^2}}$$

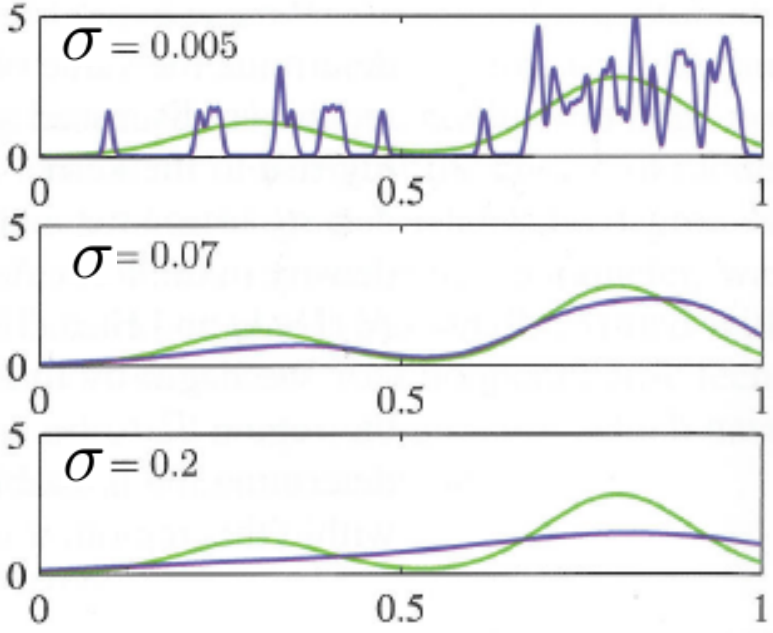
Parzen estimator methods

- Density estimation for various values of σ



Nearest neighbour methods

- A problem with Parzen estimator methods is that the parameter governing the kernel width (h or σ) is fixed for all values $\overrightarrow{x}$
  - In regions with high data density, a high kernel width might lead to over-smoothing
  - In regions with low data density, the same width may lead to noisy estimates

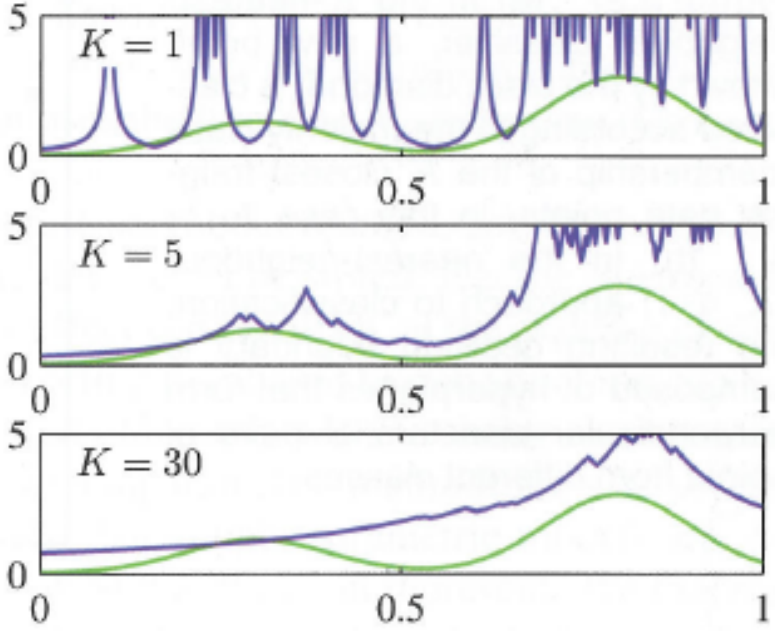


Nearest neighbour methods

- Nearest neighbour methods address this problem by adapting the width to the data density
  - Parzen estimator methods fix V and determine K from the data
  - Nearest neighbour methods fix K and choose V accordingly
- Again, we consider a point $\overrightarrow{x}$ and estimate the density $p(\overrightarrow{x})$
- The radius of the sphere is increased until K data points (the nearest neighbours) are covered

Nearest neighbour methods

- The value K then controls the amount of smoothing
- Again, an optimum value for K exists



Nearest neighbour methods

- For classification we apply the KNN-density estimation for each class
- Then, we utilise the Bayes theorem
- Assume a data set of N points with N_k points in class C_k
- In order to classify a sample $\overrightarrow{x}$, we draw a sphere containing K points around $\overrightarrow{x}$
- The sphere can contain other points regardless of their class
- Assume that the sphere has volume V and contains K_k points from class C_k

Nearest neighbour methods

- Assume: Sphere of volume V contains K_k points from class C_k
- We estimate the density of class C_k as

$$p(\vec{x}|C_k) = \frac{K_k}{N_k V}$$

- The unconditional density is given as

$$p(\vec{x}) = \frac{K}{NV}$$

- The probability to experience a class C_k is given as

$$p(C_k) = \frac{N_k}{N}$$

- With the Bayes theorem we can combine this to achieve

$$p(C_k|\vec{x}) = \frac{p(\vec{x}|C_k)p(C_k)}{p(\vec{x})} = \frac{K_k}{K}$$

Nearest neighbour methods

$$p(C_k|\vec{x}) = \frac{p(\vec{x}|C_k)p(C_k)}{p(\vec{x})} = \frac{K_k}{K}$$

- To minimise the probability of misclassification, assign $\vec{x}$ to class with the largest probability
- This corresponds to the largest value of

$$\frac{K_k}{K}$$

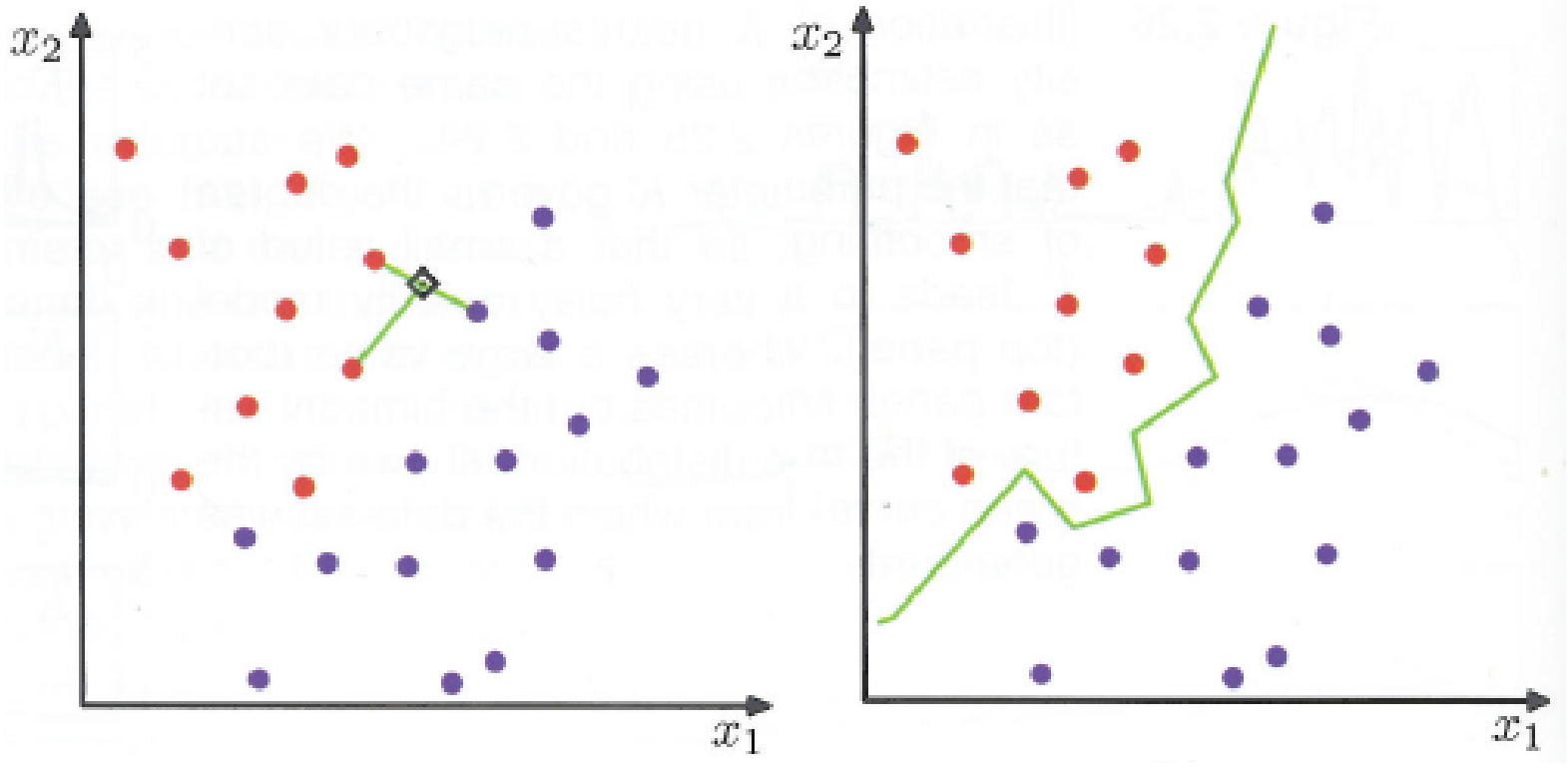
Nearest neighbour methods

- To classify a point, we identify the K nearest points
- And assign the point to the class having most representatives in this set
- The choice $K = 1$ is called the nearest neighbour rule
  - For this choice, the error rate is never more than twice the minimum achievable error rate of an optimum classifier[1]

[1] T. Cover and P. Hart: Nearest neighbour pattern classification. IEEE Transactions on Information Theory, IT-11,

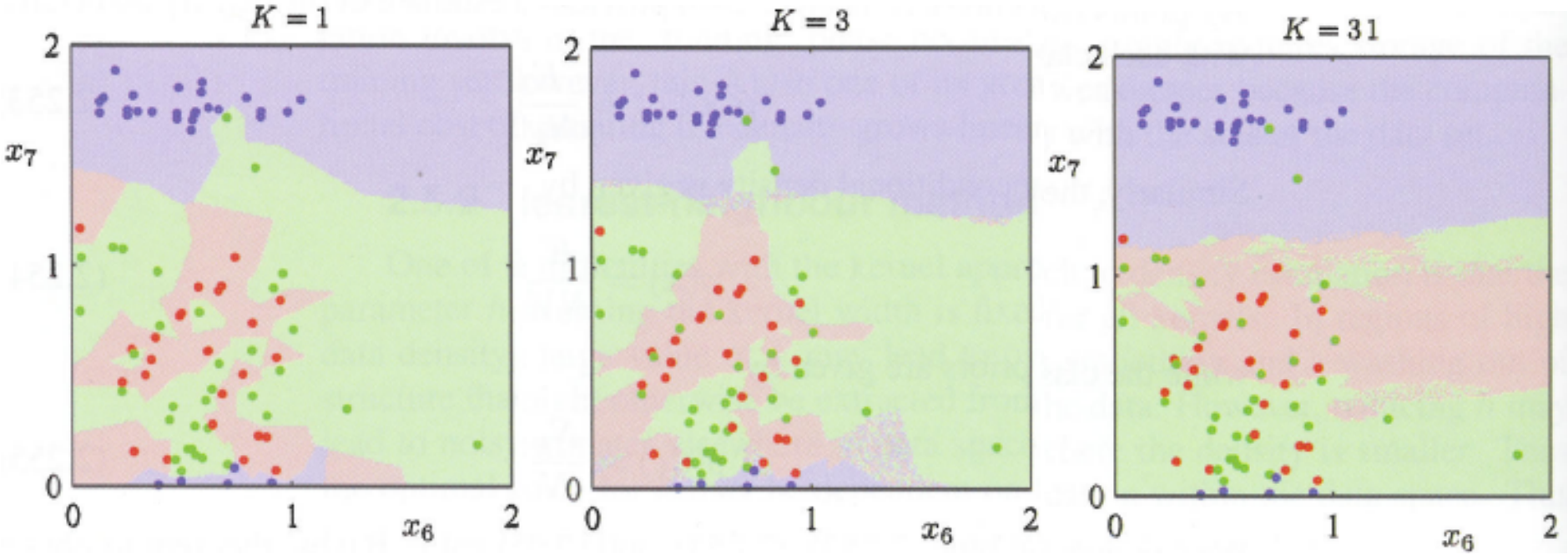
Nearest neighbour methods

- Classification of points by the K-nearest neighbour classifier



Nearest neighbour methods

- Classification of points by the K-nearest neighbour classifier



Nearest neighbour methods

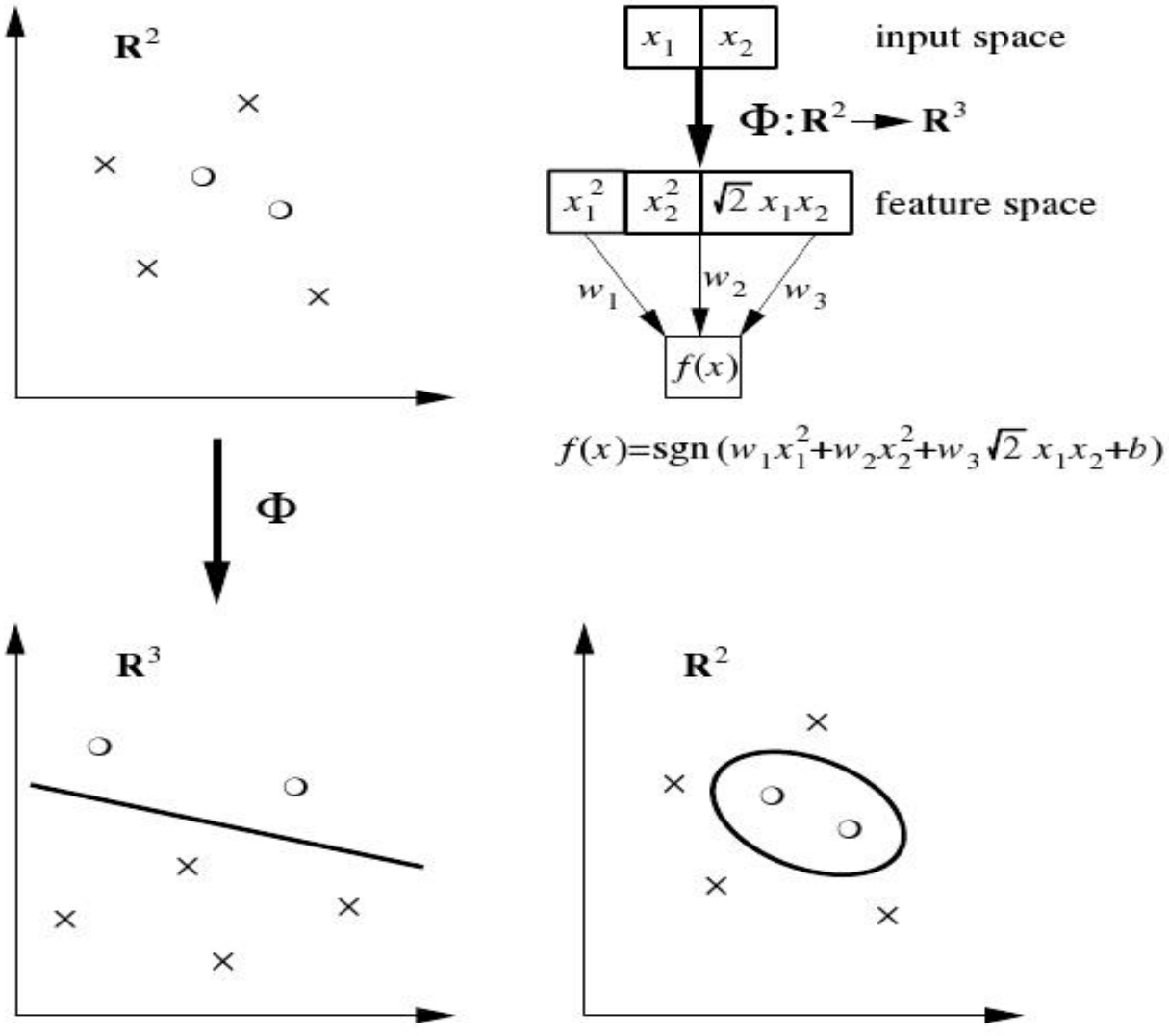
- The KNN-method and the Parzen-method are not well suited for large data sets since they require the entire data set to be stored

Outline

Support vector machines

- In classification problems we want to take an input $\vec{x}$ and assign it to one of K discrete classes C_k
- The input is divided by decision boundaries
- Here we assume that the decision boundaries are linear functions of $\vec{x}$
- Data sets that can be separated exactly by linear decision surfaces are linear separable
- With sufficiently high dimension, a data set of two classes is always linearly separable

Support vector machines



Support vector machines

- A support vector machine pre-processes data to represent patterns in a high dimension
- Dimension often much higher than original feature space
- Then, insert hyperplane in order to separate the data

Support vector machines

- A pattern $\overrightarrow{x_k}$ is transformed to $\overrightarrow{y_k} = \varphi(\overrightarrow{x_k})$
- Also, each $\overrightarrow{x_k}$ is associated with $z_k \in \{-1, 1\}$
- A linear discriminant in an augmented $\overrightarrow{y}$ space is $g(\overrightarrow{y}) = \overrightarrow{a}^t \overrightarrow{y}$
- A separating hyperplane ensures for $y_0 = 1, a_0 \geq 1$

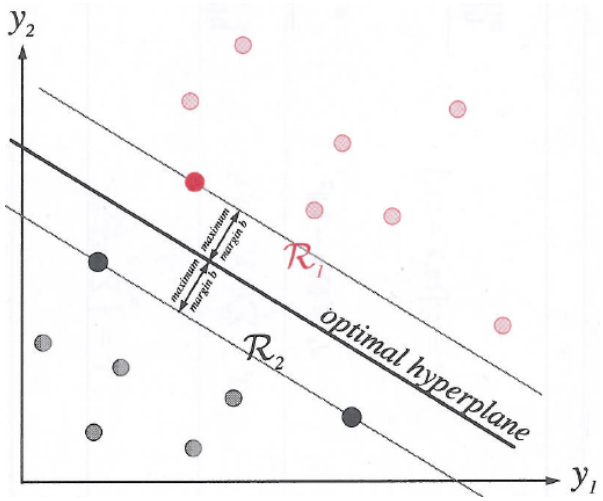
$$z_k g(y_k) \geq 1$$

Support vector machines

- The goal for support vector machines is to find a separating hyperplane with the largest margin b to the outer points in all sets

$$\frac{z_k g(y_k)}{||\overrightarrow{a}||} \geq b, \;\; k = 1, \ldots, n$$

- If no such hyperplane exists, map all points into a higher dimensional space until such a plane exists
- Support vectors satisfy '$\cdot = b$'

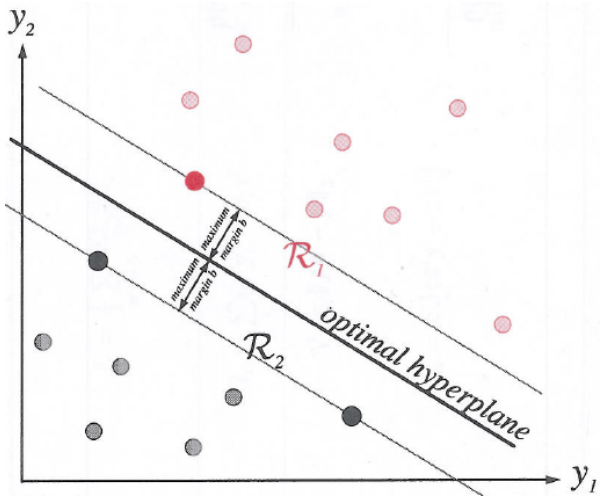


Support vector machines

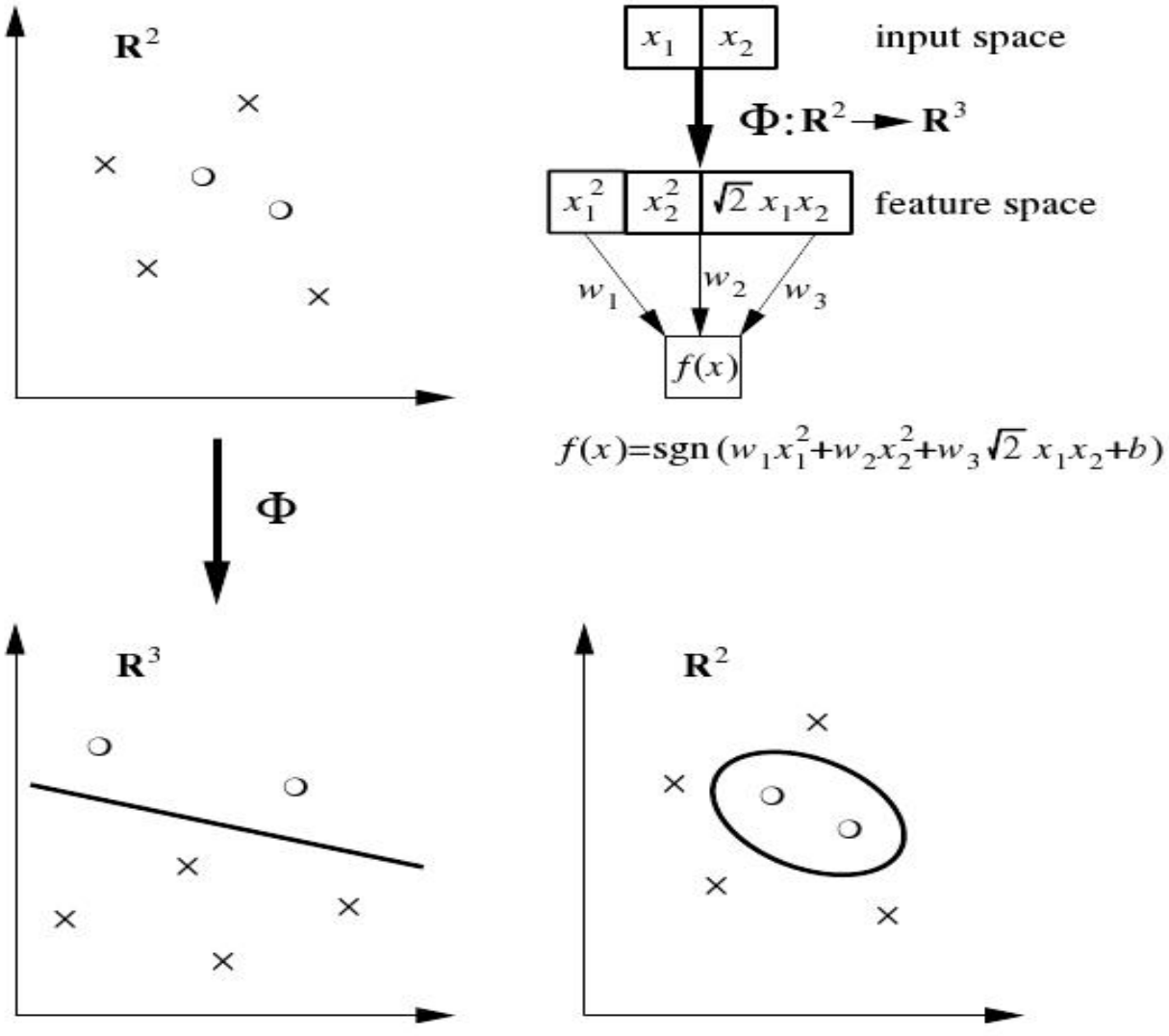
- Simple application to several classes by iterative approach:
  - belongs to class 1 or not?
  - belongs to class 2 or not?
  - ...
- Search for optimum mapping between input space and feature space complicated (no optimum approach known)

Support vector machines

- Simple learning approach to find the correct hyperplane:
  - Starting from an initial separating hyperplane
  - Find worst classified pattern (on the wrong side of the hyperplane)
  - Design a new hyperplane with this pattern as one of the support vectors
  - Iterate until all patterns are correctly classified



Support vector machines

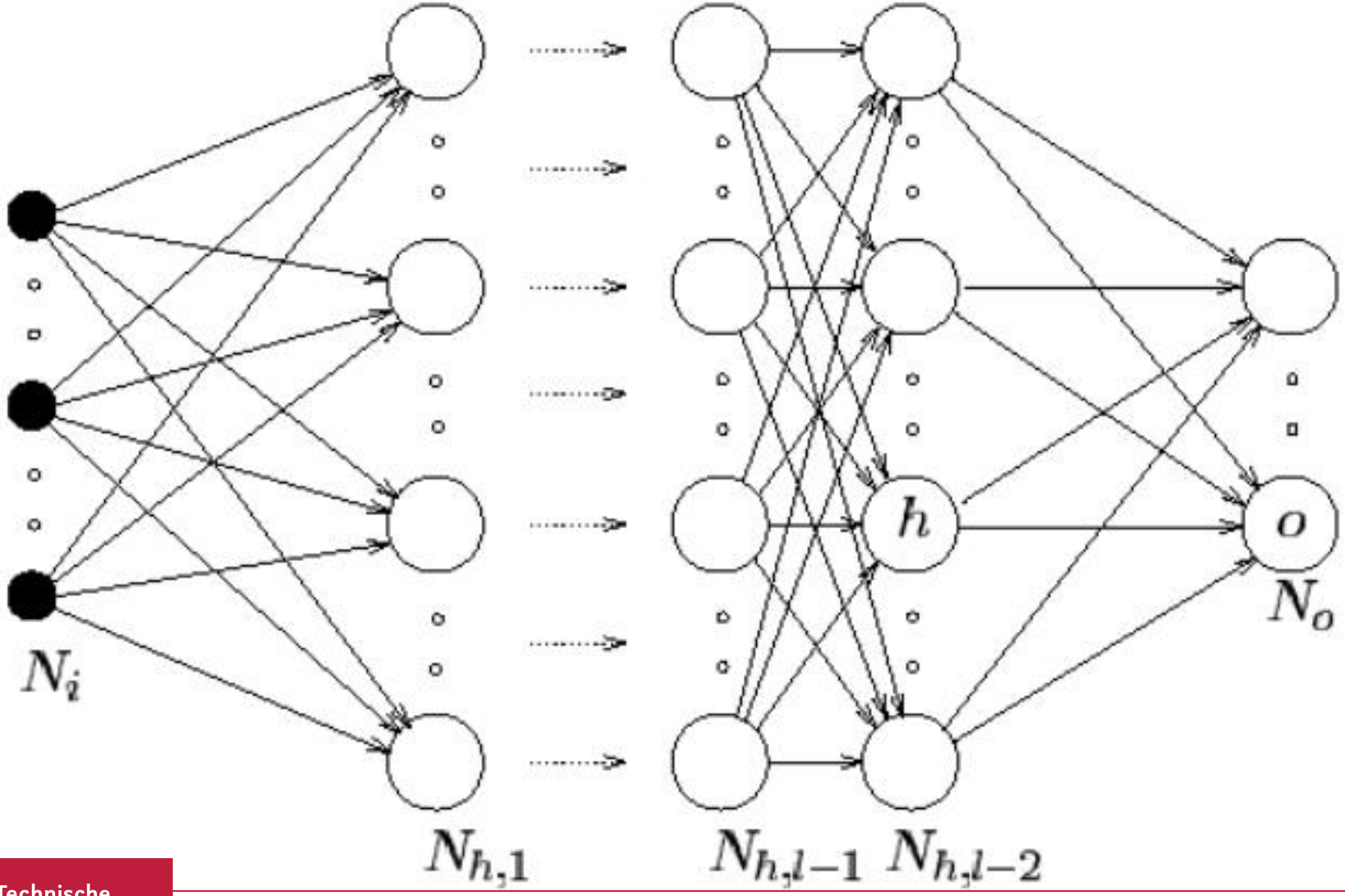


Outline

Neural networks

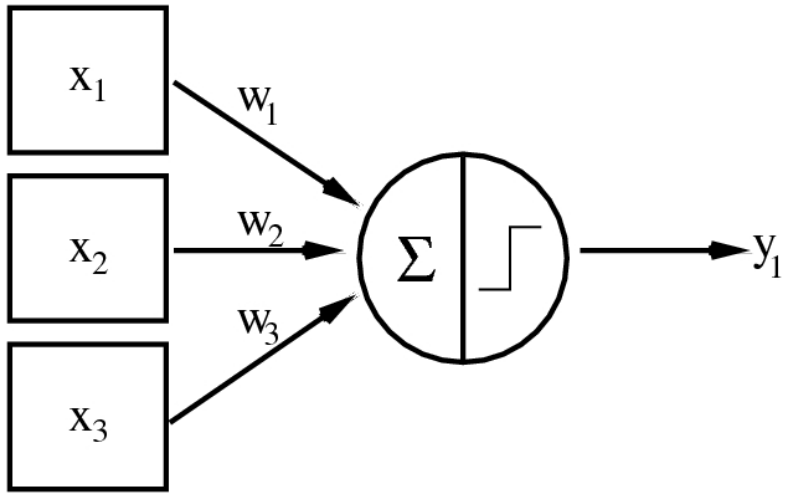
- Neural networks
  - Learn the correct mapping from an input vector to an output vector
  - Representation of the mapping function by an edge-weighted graph
  - Distinction between
    - Input neurons
    - Output neurons
    - Hidden nodes

Neural networks



Neural networks

- Neural networks
  - Input neurons are only equipped with outgoing edges
  - Hidden nodes ‘fire‘ (assign the value 1 to their outgoing edges) with respect to the configuration of the weighted ingoing edges and a threshold function Θ



$$y_i = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} x_i w_i \geq \Theta \\ 0, & \text{else.} \end{cases}$$

Neural networks

Learning with back-propagation (schematic):

- Iterate until the error is sufficiently small
  1. Choose a training-pair and copy it to the input layer
  2. Propagate it through the network
  3. Calculate error between computed and expected output
  4. Propagate the sum product of the weights back into the network in order to calculate the error in internal layers
  5. Adapt weights to the error

Neural networks

Discussion

- With only one layer of hidden nodes it is possible to represent arbitrary multi-dimensional functions
- Is well suited to work on noisy input data
- Implicit clustering of input data possible
- Issues:
  - Complicated to extend existing network (e.g. when new features are added)

Neural networks

- A Neural network consists of a series of functional transformations
- For the input layer, we construct M linear combinations of the input variables $x_1, \ldots, x_D$ and weights $w_1, \ldots, w_D$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

- Each a_j is transformed using a differentiable, non-linear activation function

$$z_j = h(a_j)$$

Neural networks

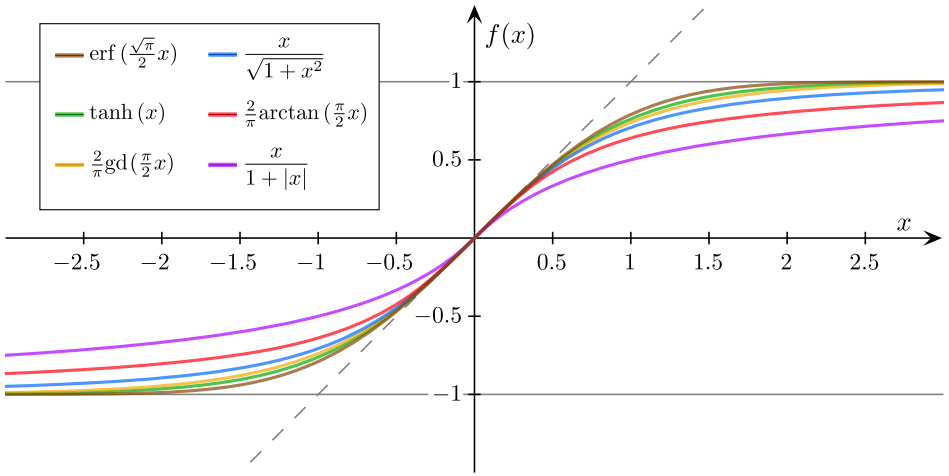
- Input layer: M linear combinations of $x_1, \ldots, x_D$ and $w_1, \ldots, w_D$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

- Differentiable, non-linear activation function:

$$z_j = h(a_j)$$

- The $h(\cdot)$ function is usually a sigmoidal function or tanh



Neural networks

- The values z_j are then again linearly combined in the hidden layers:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

- with $k = 1, \ldots, K$ describing the total number of outputs
- Again, these values are transformed using a sufficient transformation function σ to obtain the network outputs y_k

$$y_k = \sigma(a_k)$$

- For multi-class problems, we use a function such as

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Neural networks

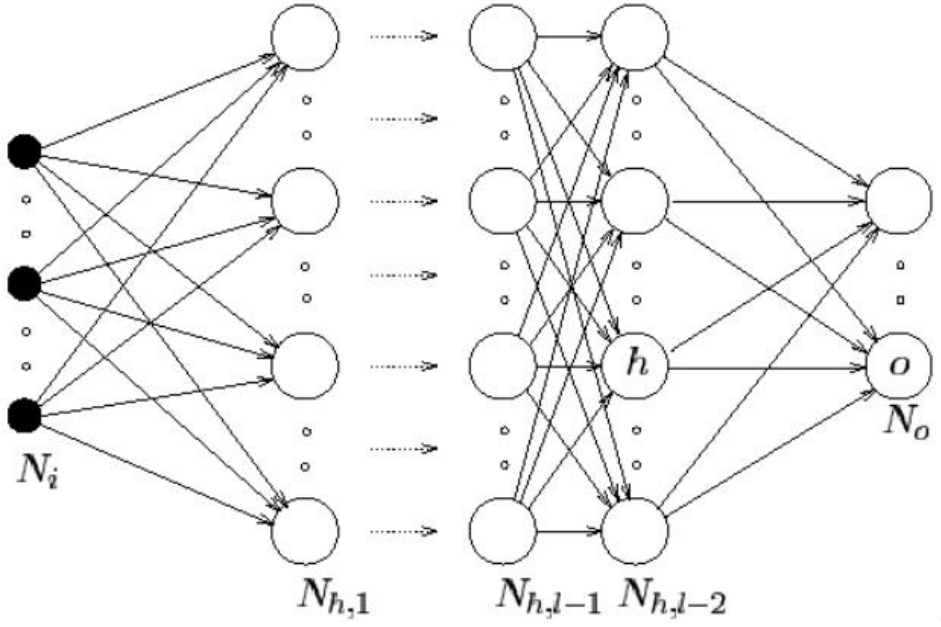
- We can combine these various stages to give the overall network function in the form

$$y_k(\vec{x}, \vec{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- When the network contains multiple hidden layers, these are added analogously

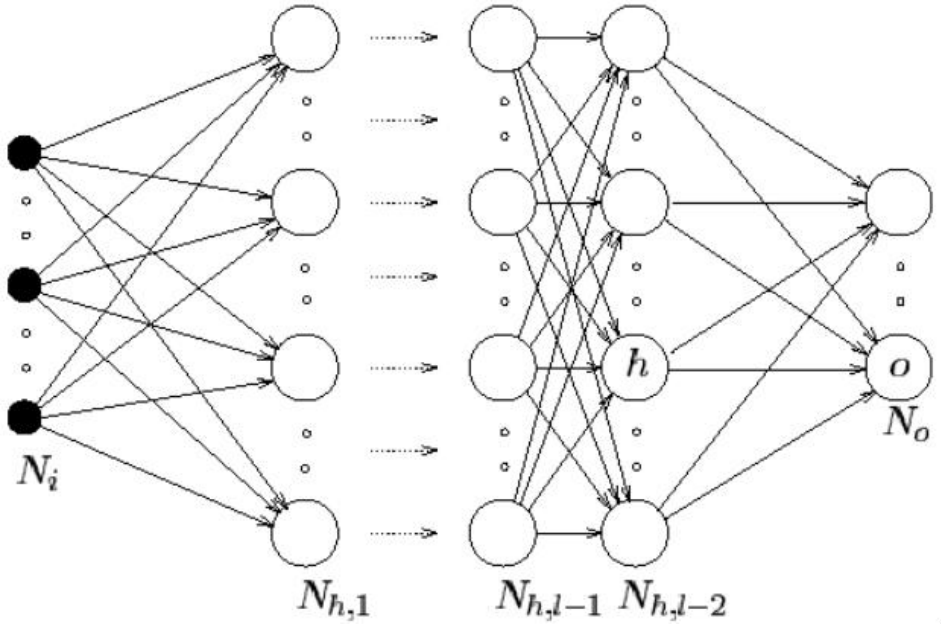
Neural networks

- When the activation functions of all hidden units in a network are linear, we can always find an equivalent network without hidden units
  - Since the composition of successive linear transformations is itself a linear transformation



Neural networks

- When the number of hidden units is smaller than the number of input or output units, not all linear functions can be generated
  - Since Information is lost in the dimensionality reduction at the hidden units



Neural networks

- Sometimes, skip-layer connections are utilised
  - In principle, a network with sigmoidal hidden units can always mimic skip layer connections
  - By using sufficiently small first-layer weight
  - and then compensating with a large weight from the hidden unit to the output

Neural networks

- Neural networks are Universal approximators[2] [3] [4] [5] [6] [7] [8] [9]
  - A two-layer network with only linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy

[2] K. Funahashi: On the approximate realisation of continuous mappings by neural networks, Neural Networks, 2(3), 183-192, 1989

[3] G. Cybenko: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2, 304-314, 1989

[4] K. Hornik, M. Sinchcombe, H. White: Multilayer feed-forward networks are universal approximators. Neural Networks, 2(5), 359-366, 1989

[5] N.E. Cotter: The stone-Weierstrass theorem and its application to neural networks. IEEE Transactions on Neural Networks 1(4), 290-295, 1990

[6] Y. Ito: Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Networks 4(3), 385-394, 1991

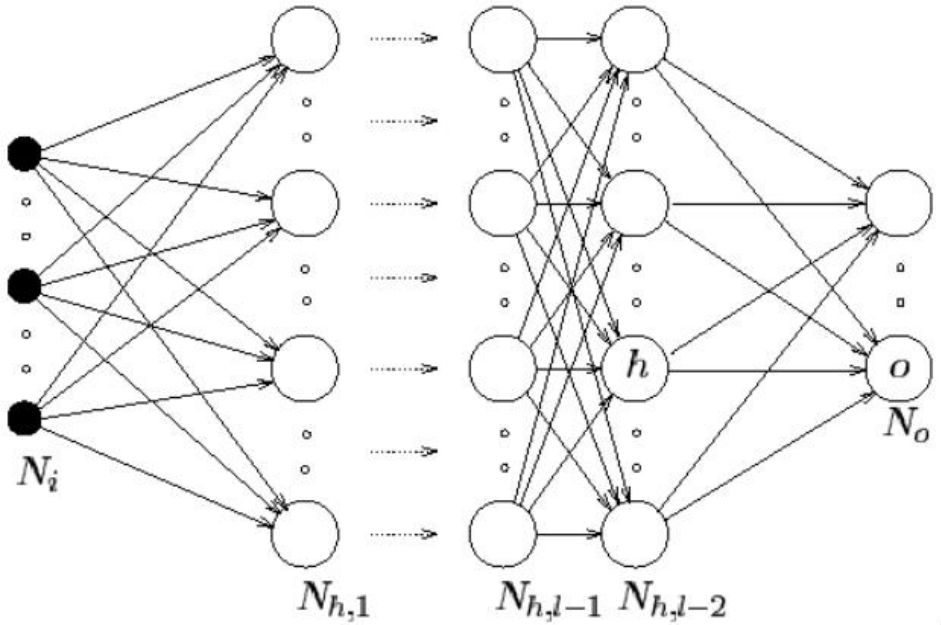
[7] K. Hornik: Approximation capabilities of multilayer feed forward networks: Neural Networks, 4(2), 251-257, 1991

[8] Y.V. Kreinovich: Arbitrary non-linearity is sufficient to represent all functions by neural networks: a theorem. Neural Networks 4(3), 381-383, 1991

[9] [illegible] Pattern Recognition and Neural Networks. Cambridge University Press, 1996

Neural networks

- Remaining issue in neural networks
  - How to find suitable parameter values given a set of training data
  - Several learning approaches have been proposed



Neural networks

- A simple approach to the problem of determining the network parameters is to minimise a sum-of-squared error function
  - Given a training set $\overrightarrow{x_n}$ with $n \in \{1, \ldots, N\}$
  - And a corresponding set of target vectors $\overrightarrow{t_n}$
  - We minimise the error function

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(\overrightarrow{x_n}, \overrightarrow{w}) - \overrightarrow{t_n})^2$$

Neural networks

Classification with neural networks

- Classification of two classes $\mathcal{C}_1$ and $\mathcal{C}_2$
  - We consider a network with a single output

$$y = \sigma(a) \equiv \frac{1}{1 + e^{-a}}$$

  - The output is interpreted as the conditional probability $p(\mathcal{C}_1|\overrightarrow{x})$
  - Analogously, we have $p(\mathcal{C}_2|\overrightarrow{x}) = 1 - p(\mathcal{C}_1|\overrightarrow{x})$

Neural networks

Classification with neural networks

- Classification of K classes $\mathcal{C}_1, \cdots, \mathcal{C}_K$
  - Binary target variables $t_k \in \{0, 1\}$
  - Network outputs are interpreted as $y_k(\vec{x}, \vec{w}) = p(t_k = 1|\vec{x})$

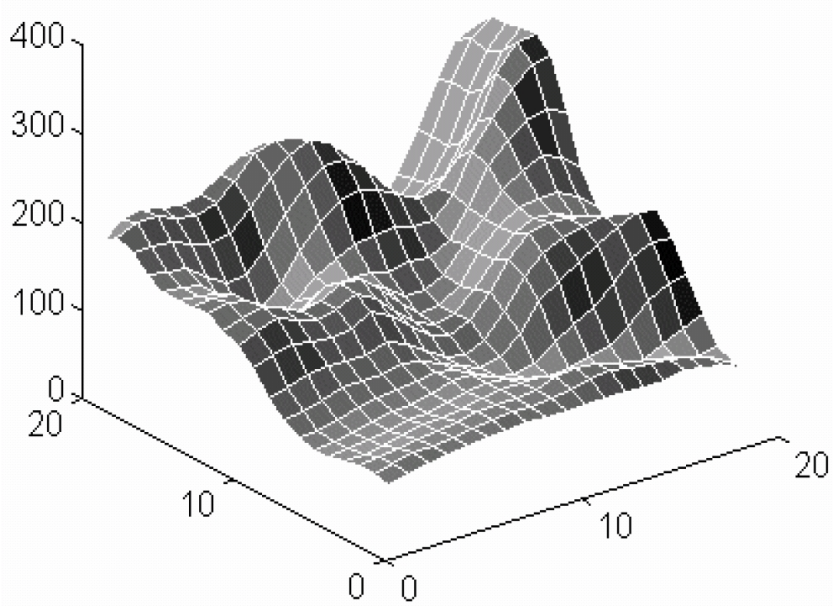
Introduction to self organising maps

- Self organising map (SOM) algorithm proposed by Teuvo Kohonen[10]
- Presented it as a model of the self-organisation of neural connections.
- Map high dimensional input data to low dimensional representation
- Based on neural network learning of the underlying mapping

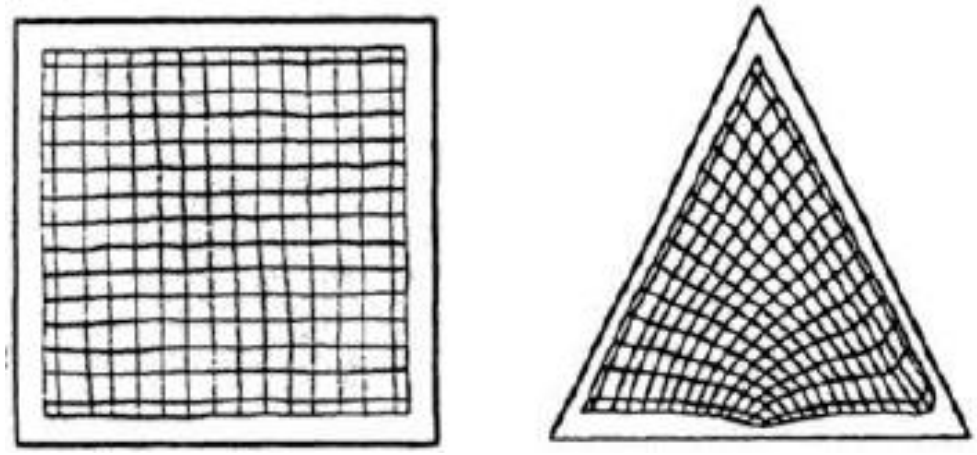
[10] T. Kohonen, *Self-Organizing Maps*, Springer, 2001.

Introduction to self organising maps

- Idea:
  - Present all points in a source space by points in a target space
  - Given a sequence of points in a sample space,
  - Create a mapping of these points into a target space that respects the neighbourhood relation in the sample space

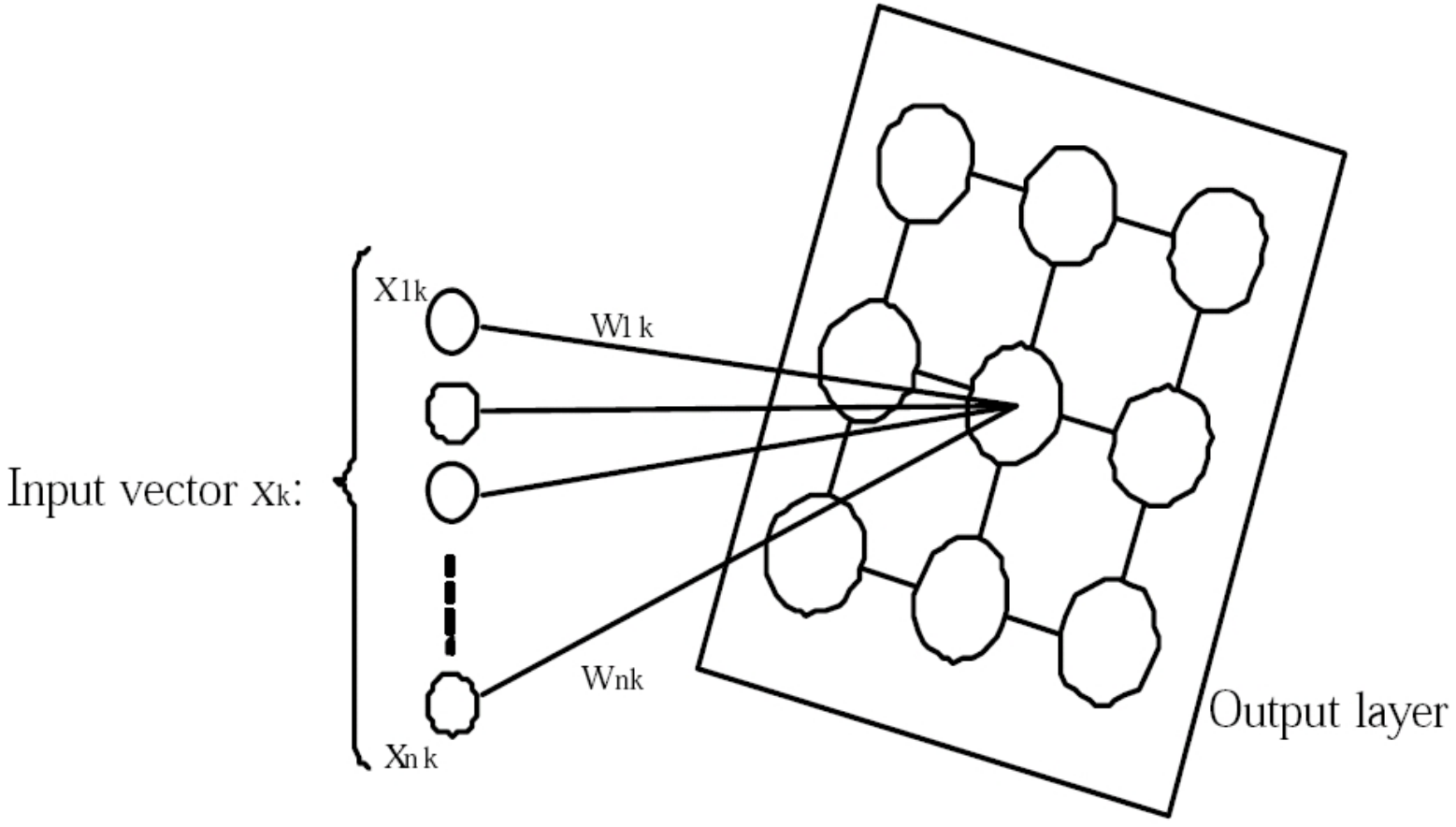


Introduction to self organising maps

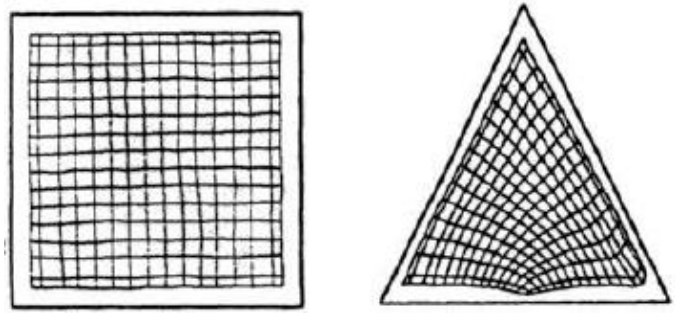


- SOM is a topology preserving lattice of a predefined number of nodes that represents a topology of elements in the input space.
- Algorithm inherits self-organisation property
  - Able to produce organisation starting from possibly total disorder.
  - SOM algorithm defines and preserves neighbourhood structure between all nodes of the map.
- Learning by two layer neural network

Introduction to self organising maps



Introduction to self organising maps

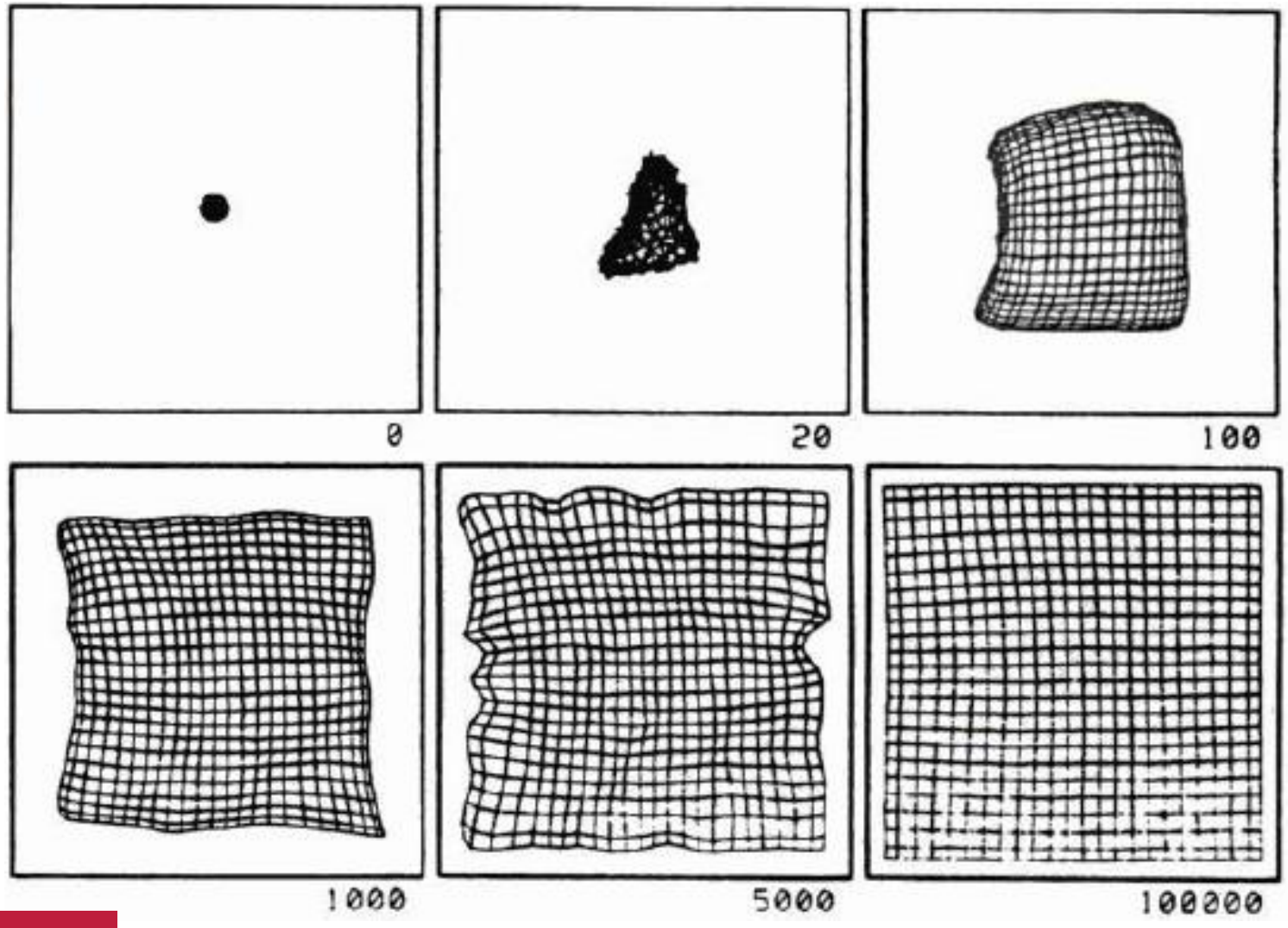


- When a pattern $\overrightarrow{\phi_i}$ is presented, each node (represented by outer neurons) in the target space computes its activation $\overrightarrow{\phi}_i^t \overrightarrow{w}$.
- The most activated node y^* and the weights to its neighbours are updated according to a learning rate $\rho(t)$

$$w_{ki}(t+1) = w_{ki}(t) + \rho(t)\Lambda(|y - y^*|)(\overrightarrow{\phi_i} - w_{ki}(t))$$

- $\Lambda(\cdot)$ defines a non-increasing neighbourhood function and $|y - y^*|$ describes the distance of nodes in the neighbourhood

SOM – Self organisation



SOM – Definition

- We recapitulate a condensed definition of the SOM algorithm that can be found in[11]

Self organising maps

- Let $I = \{\overrightarrow{\eta_1}, \ldots, \overrightarrow{\eta_{|S|}}\}$ be a set of km-dimensional vectors that are associated with nodes in a lattice.
- Neighbourhood structure provided by symmetrical, non-increasing neighbourhood function $d : I \times I \to \mathbb{R}$ which depends on the distance between two nodes $\overrightarrow{\eta_i}$ and $\overrightarrow{\eta_j} \in I$.
- The state of the map at time t is given by

$$\eta(t) = \left(\overrightarrow{\eta_1(t)}, \overrightarrow{\eta_2(t)}, \ldots, \overrightarrow{\eta_{|S|}(t)}\right),$$

[11] M. Cottrell, J.C. Fort and G. Pages, *Theoretical aspects of the SOM algorithm*, Neurocomputing, pp. 119-138, vol

SOM – Definition

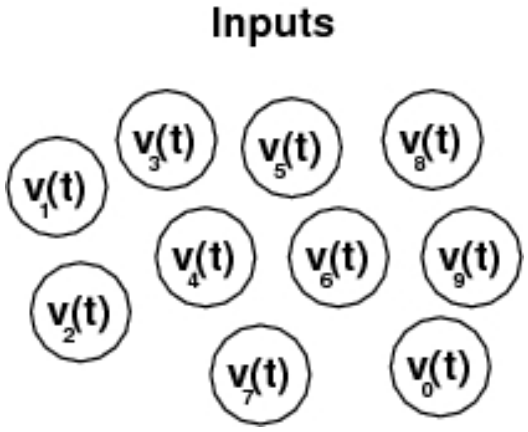
Self organising map algorithm

The SOM algorithm is recursively defined by

$$
\begin{aligned}
i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right) &= \mathit{argmin}\left\{\left\|\overrightarrow{v(t+1)} - \overrightarrow{\eta_i(t)}\right\|, \overrightarrow{\eta_i(t)} \in \eta(t)\right\}, \\
\overrightarrow{\eta_i(t+1)} &= \overrightarrow{\eta_i(t)} - \varepsilon_t d\left[i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right), \overrightarrow{\eta_i}\right] \\
&\quad \cdot \left(\overrightarrow{\eta_i(t)} - \overrightarrow{v(t+1)}\right), \forall \overrightarrow{\eta_i} \in I.
\end{aligned}
$$

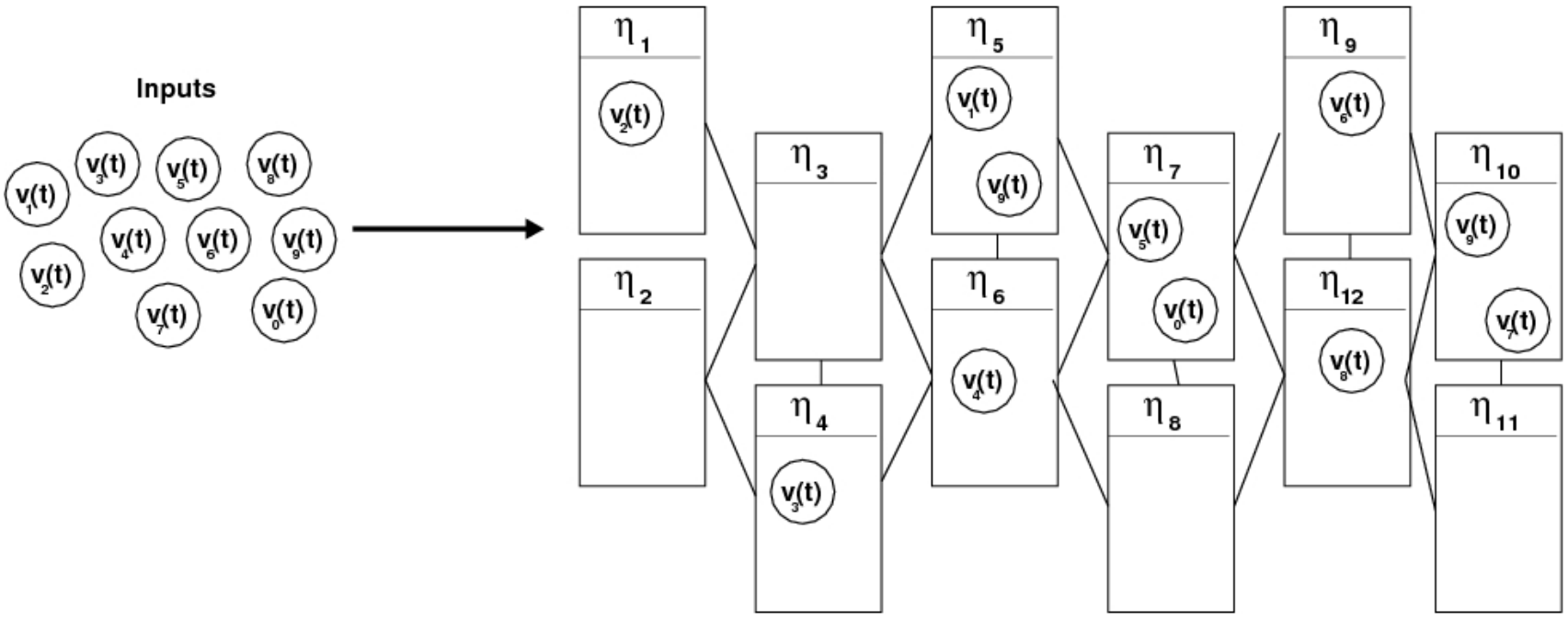
- In this formula, $i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right)$ corresponds to the node in the network that is closest to the input vector.
- Parameter ε_t controls the adaptability of the self organising map.

SOM – Operational principle



- Input values $v_i(t)$ are to be mapped onto the target space

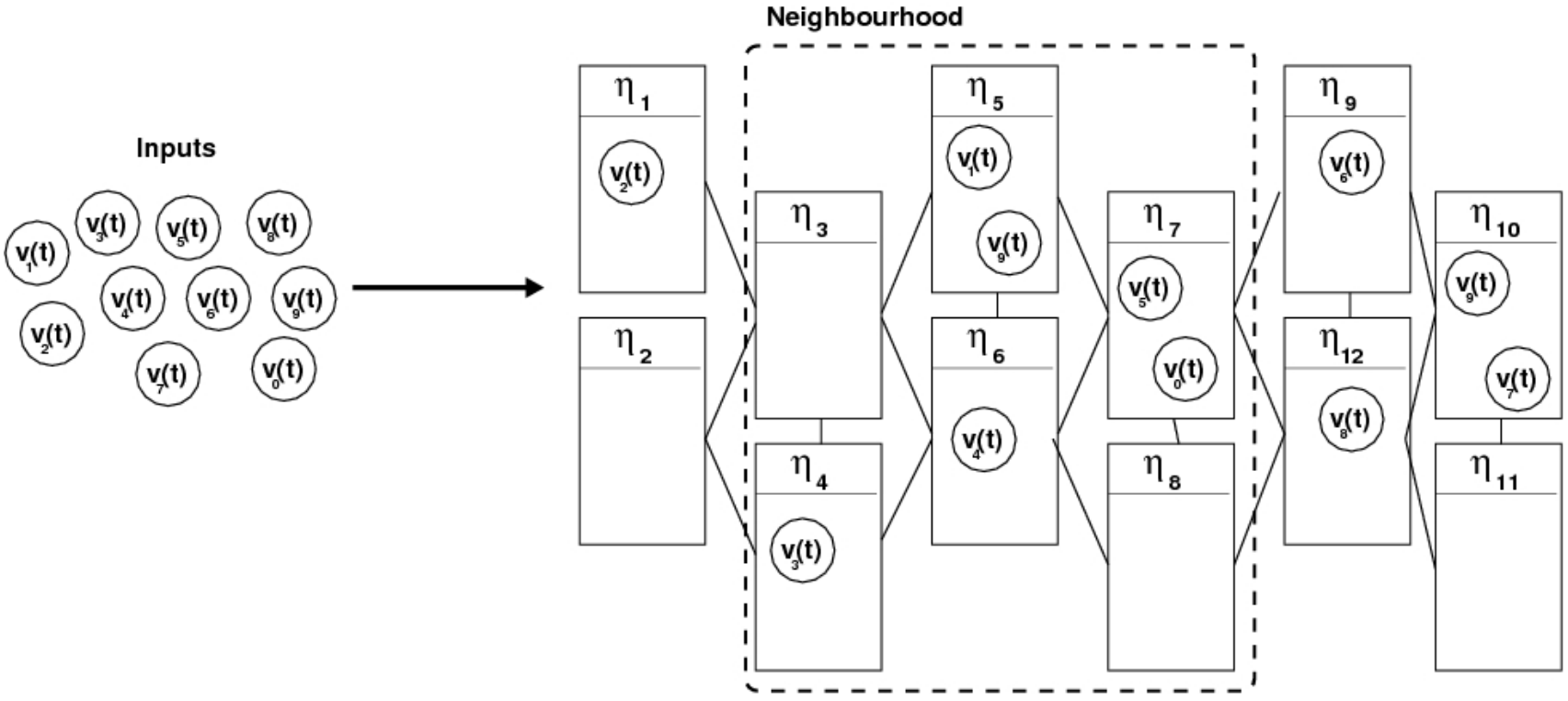
SOM – Operational principle



- The node with the lowest distance is associated with the input value:

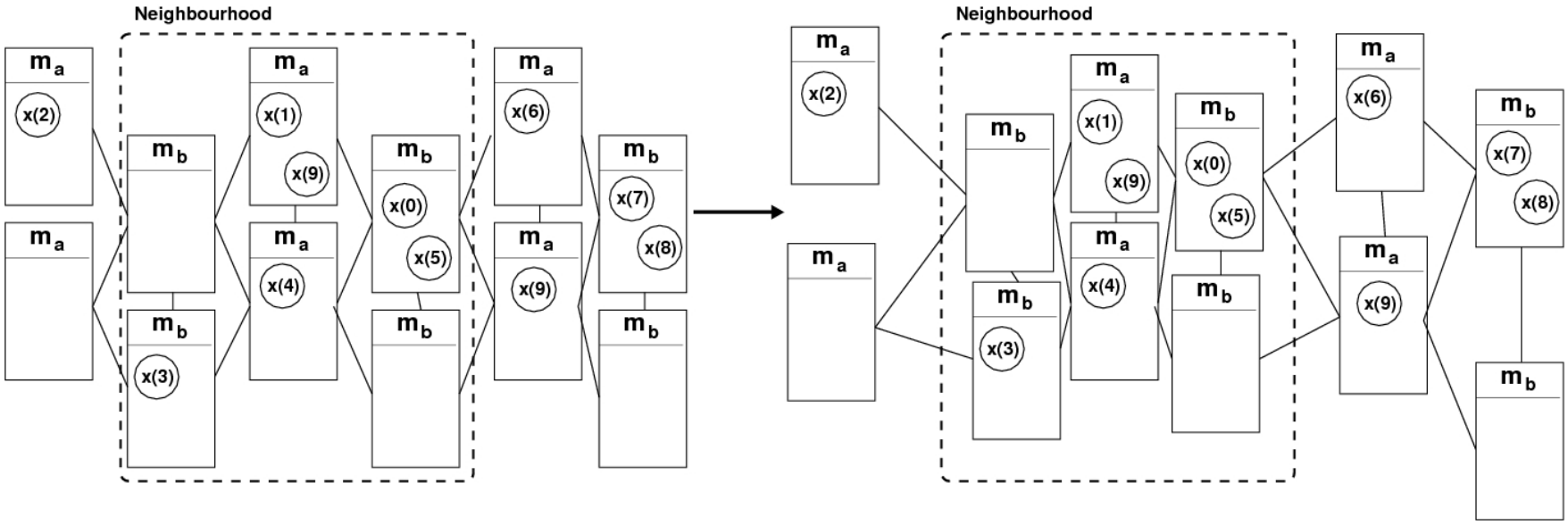
$$i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right) = argmin\left\{\left\|\overrightarrow{v(t+1)} - \overrightarrow{\eta_i(t)}\right\|, \overrightarrow{\eta_i(t)} \in \eta(t)\right\}$$

SOM – Operational principle



- Nodes in the neighbourhood of the associated node are moved closer to the input value

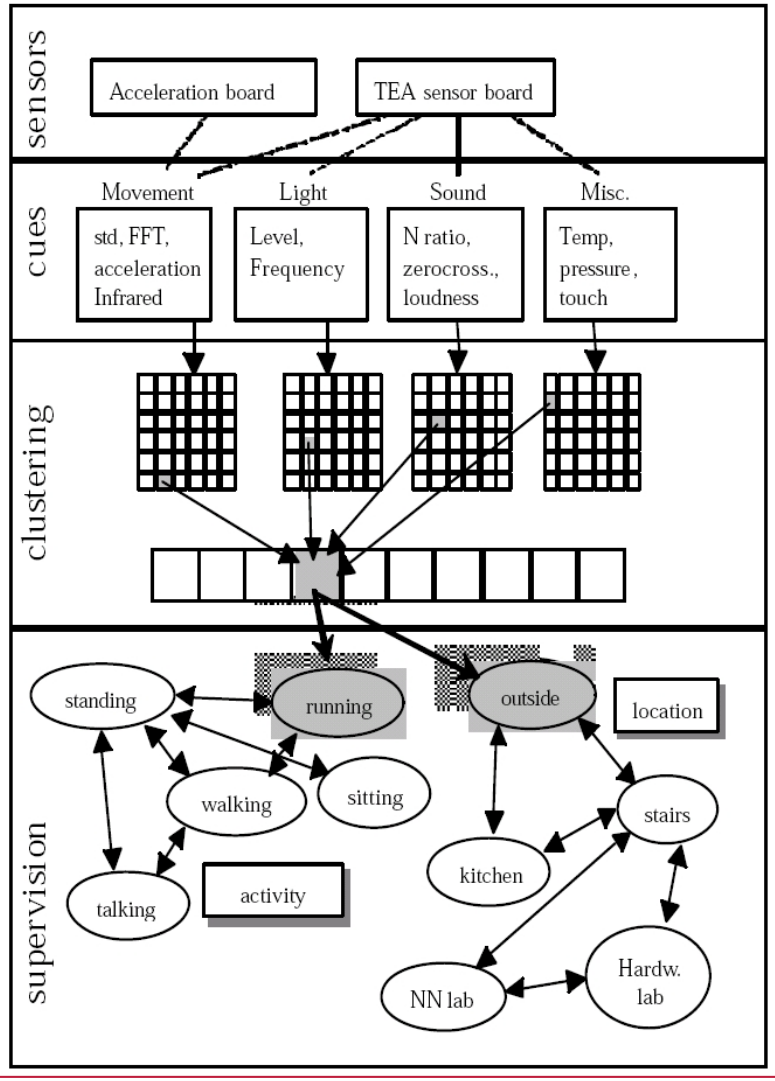
SOM – Operational principle



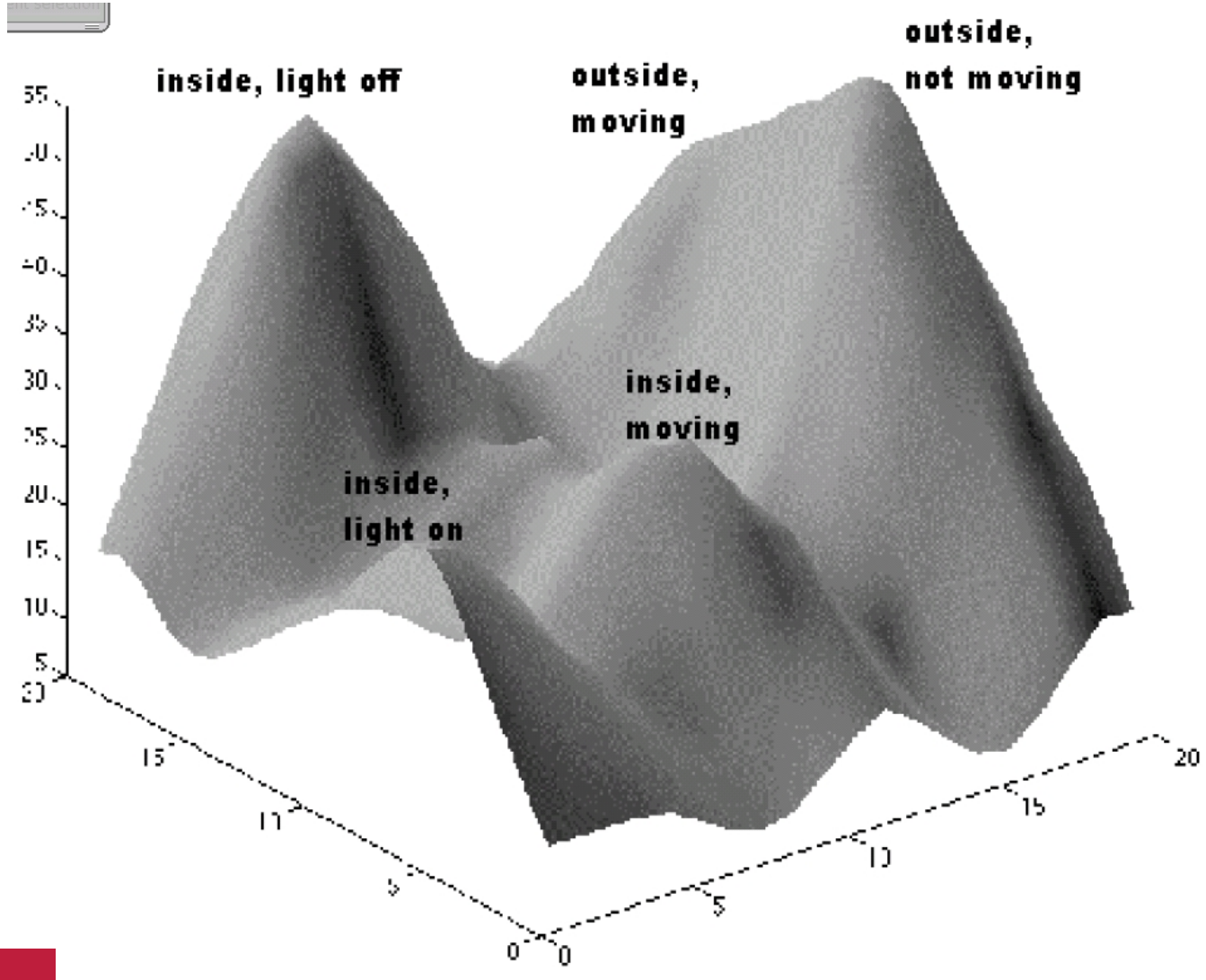
- Nodes in the neighbourhood of the associated node are moved to the input value

$$\overrightarrow{\eta_i(t+1)} = \overrightarrow{\eta_i(t)} - \varepsilon_t d\left[i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right), \overrightarrow{\eta_i}\right] \cdot \left(\overrightarrow{\eta_i(t)} - \overrightarrow{v(t+1)}\right), \forall \overrightarrow{\eta_i} \in I.$$

SOM – Example application: TEA



SOM – Example application: TEA



SOM – Remarks

- It has been proven that the SOM algorithm always converges[12]
- Normalisation of input vectors might improve numerical accuracy
- Not guaranteed that self-optimisation will always occur (Dependent on choice of parameters)
- The SOM is not optimising any well-defined function, so that it is difficult to set the parameters of the model [13]

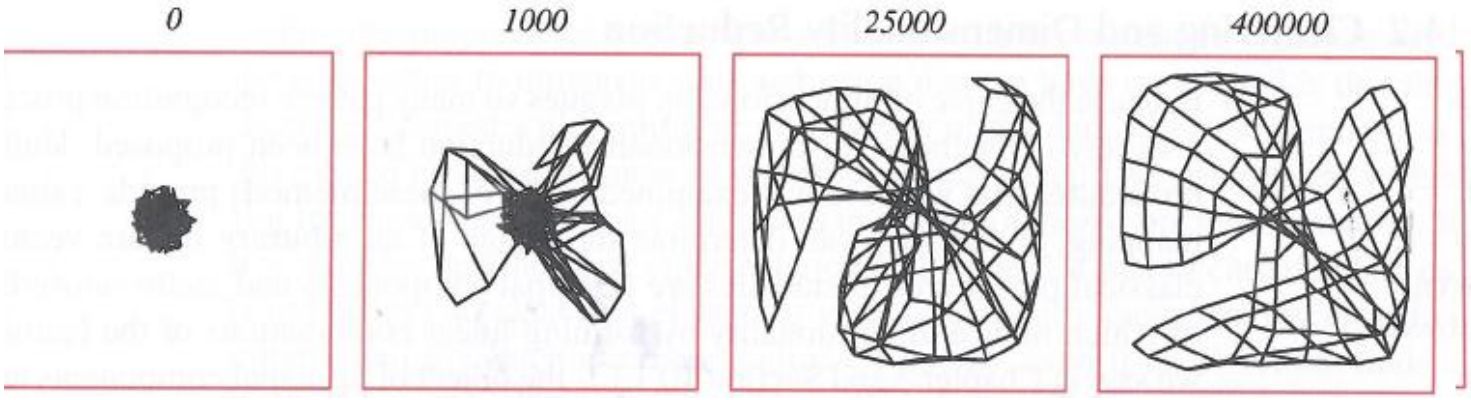
[12] Y. Cheng, *Neural Computation*, 9(8), 1997.

[13] E. Erwin, K. Obermayer, K. Schulten: Self-organising maps: Ordering, convergence properties and energy functions. Biological Cybernetics, 67, 47-55, 1992

Problems of SOMs

- If the neighbourhood is chosen to be too small, the map will not be ordered globally.

Problems of SOMs



- The map created as target space might have several orientations
- It is possible that one part of the map is created following one orientation, while other parts are created following other orientations.

Software tools for SOMs – The SOM_PAK

SOM_PAK First public-domain software package (1990)[14]

- Released by Laboratory of Computer and Information Science of Helsinki University of Technology
- Source code (ANSI C) and documentation completely available
- Available for UNIX or MS DOS
- Features:
  - Standard incremental-learning SOM
  - (simple) graphics programs included
  - Map size and vector dimension not restricted
  - Several neighbourhood functions available
  - Able to handle largest scale problems

[14] ...is.hut.fi/research/software.shtml

Software tools for SOMs – The SOM Toolbox

SOM Toolbox Toolbox for MatLab (1996)[15]

- Released by Laboratory of Computer and Information Science of Helsinki University of Technology
- MatLab version 5 or higher required
- Slower than SOM_PAK
- Features:
  - Standard incremental-learning SOM and Batch Map SOM
  - Map size and vector dimension not restricted
  - Neighbourhood and training sequences identical to SOM_PAK
  - Improved visualisation and analysis capabilities

[15] ...is.hut.fi/software.shtml

Software tools for SOMs – The Neural Networks Tool

Nenet User friendly ANN Toolbox (1997)[16]

- Released by the Neural network team of Helsinki University of Technology
- 32-bit Windows 95/NT recommended
- Suited for small scale problems only
- Features:
  - Standard incremental SOM
  - Several visualisation options:
    - Component planes with trajectories
    - U-matrix
    - 3D hit histograms
    - Display of active neuron coordinates
  - Easy to use; Good graphics programs

[16] ...nbnet.fi/~phodju/nenet/Nenet/Gneral.html

Software tools for SOMs – Viscovery SOMine

Viscovery SOMine Commercial SOM software package [17]

- Released by Eudaptics GmbH in Austria
- Windows NT and Windows NT 4.0
- Features:
  - User-friendly, flexible and powerful package
  - Interfaces for GUI, OLE, SQL and DB2
  - Batch Map algorithm
  - High computing speed
  - Unlimited map size and vector dimension
  - Several visualisation options:
    - Component planes with trajectories
    - U-matrix
    - Cluster windows
    - Iso-contours of hit density

[17] http://www.eudaptics.co.at

Outline

Markov chains

- Markov processes
  - Intensively studied
  - Major branch in the theory of stochastic processes
- A. A. Markov (1856 – 1922)
- Extended by A. Kolmogorov by chains of infinitely many states
  - 'Anfangsgründe der Theorie der Markoffschen Ketten mit unendlich vielen möglichen Zuständen' (1936) [18]

[18] A. Kolmogorov, *Anfangsgründe der Theorie der Markoffschen Ketten mit unendlich vielen möglichen Zuständen*, 1936.

Markov chains

- Markov chains
  - Theory of Markov chains applied to a variety of algorithmic problems
  - Standard tool in many probabilistic applications
- Intuitive graphical representation
  - Possible to illustrate properties of stochastic processes graphically
- Popular for their simplicity and easy applicability to huge set of problems[19]

[19] William Feller, *An introduction to probability theory and its applications*, Wiley, 1968.

Markov chains

- Independent trials of events
  - Set of possible outcomes of a measurement E_i associated with occurrence probability p_i
  - Probability to observe sample sequence:
    - $P\{(E_1, E_2, \ldots, E_i)\} = p_1 p_2 \cdots p_i$
- Dependent trials of events
  - Probability to observe specific sequence $E_1, E_2, \ldots, E_i$ obtained by conditional probability:

$$P(E_i|E_1, E_2, \ldots, E_{i-1})$$

  - In general:

$$P(E_i|E_1, E_2, \ldots, E_{i-1}) \neq P(E_i|E_2, E_1, E_3, E_4, \ldots, E_{i-1})$$

Markov chains

- Independent random variables
  - Number of coin tosses until 'head' is observed
  - Radioactive atoms always have the same probability of decaying at the next trial
- Dependent random variables
  - The knowledge that no car has passed for five minutes increases our expectation that it will come soon.
  - Coin tossing:
    - Probability that the cumulative numbers of heads and tails will equalize at the second trial is $\frac{1}{2}$
    - Given that they did not, the probability that they equalize after two additional trials is only $\frac{1}{4}$

Markov chains

Markov property

In the theory of stochastic processes the described lack of memory is connected with the Markov property.

- Theory of Markov chains:
  - Outcome of any trial depends exclusively on the outcome of the directly preceding trial
  - Outcome of E_k is no longer associated with fixed probability p_k
    - Instead: With every pair (E_i, E_j) a conditional probability p_{ij}
    - Probability that E_j is observed after E_i
    - Additionally: Probability a_i of the event E_i

Markov chains

Markov chain

A sequence of observations $E_1, E_2, \ldots$ is called a Markov chain if the probabilities of sample sequences are defined by

$$P(E_1, E_2, \ldots, E_i) = a_1 \cdot p_{12} \cdot p_{23} \cdot \ldots \cdot p_{(i-1)i}.$$

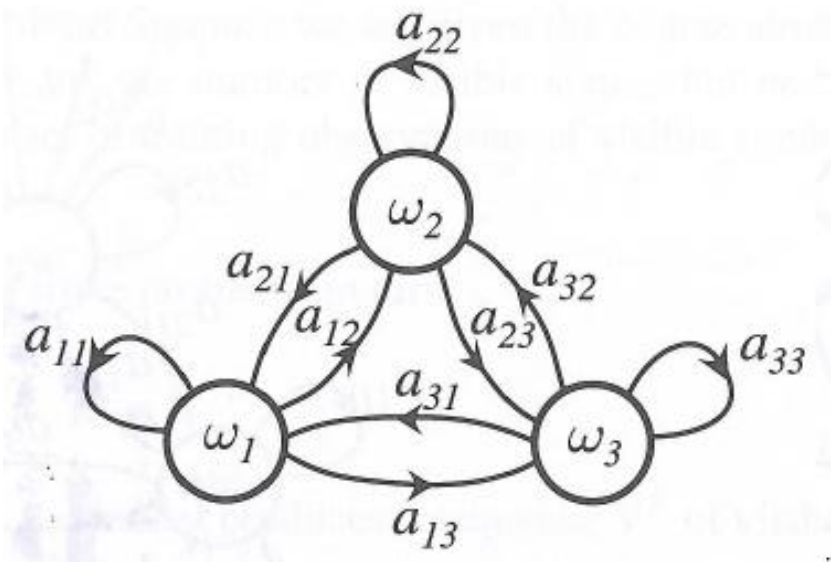
and fixed conditional probabilities p_{ij} that the event E_i is observed directly in advance of E_j.

Markov chains

- Theory of Markov chains:
  - $P\{(E_i, E_j)\} = a_i p_{ij}$
  - $P\{(E_i, E_j, E_k)\} = a_i p_{ij} p_{jk}$
  - $P\{(E_i, E_j, E_k, E_l)\} = a_i p_{ij} p_{jk} p_{kl}$
  - $P\{(E_i, E_j, \ldots, E_m, E_n)\} = a_i p_{ij} \ldots p_{mn}$

Markov chains

- Markov chain sometimes modelled as directed graph $G = (V, E)$
- Labelled edges in E
- states (or vertices) in V.
- Transition probabilities p_{ij} between $E_i, E_j \in V$



Markov chains

- Markov chain described by probability a for initial distribution and matrix P of transition probabilities.

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots \\ p_{21} & p_{22} & p_{23} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

- P is a square matrix with non-negative entries that sum to 1 in each row.
- P is called a stochastic matrix

Markov chains

- p_{ij}^k denotes probability that E_j is observed exactly k observations after E_i was observed.
- Calculated as the sum of the probabilities for all possible paths $E_i E_{i_1} \cdots E_{i_{k-1}} E_j$ of length k
- We already know

$$p_{ij}^1 = p_{ij}$$

- Consequently:

$$\begin{aligned} P_{ij}^2 &= \sum_\nu p_{i\nu} \cdot p_{\nu j} \\ P_{ij}^3 &= \sum_\nu p_{i\nu} \cdot p_{\nu j}^2 \end{aligned}$$

Markov chains

- By mathematical induction:

$$p_{ij}^{n+1} = \sum_{\nu} p_{i\nu} \cdot p_{\nu j}^{n}$$

- and

$$p_{ij}^{n+m} = \sum_{\nu} p_{i\nu}^{m} \cdot p_{\nu j}^{n} = \sum_{\nu} p_{i\nu}^{n} \cdot p_{\nu j}^{m}$$

Markov chains

- Similar to matrix P we can create a matrix P^n that contains all p_{ij}^n
- We obtain P_{ij}^{n+1} from P^{n+1} by multiplying all elements of the i-th row of P with the corresponding elements of the j-th column of P^n and add all products.
- Symbolically: $P^{n+m} = P^n P^m$.

Markov chains

Closed set of states

- A set C of states is closed if no state outside C can be reached from any state E_i in C.
- For an arbitrary set C of states the smallest closest set containing C is called the closure of C
- A single state E_k forming a closed set is called absorbing

Markov chains

Closed sets in stochastic matrices

If in a matrix P^n all rows and all columns corresponding to states outside a closed set C are deleted, the remaining matrices are again stochastic matrices.

Markov chains

Irreducible Markov chain

A Markov chain is irreducible if there exists no closed set other than the set of all states.

Criterion for irreducible chains

A chain is irreducible if, and only if, every state can be reached from every other state.

Markov chains

Periodicity of states

- The state E_j has period $t > 1$ if $p_{jj}^n = 0$ unless $n = vt$ is a multiple of t and t is the largest integer with this property.
- The state E_j is aperiodic if no such $t > 1$ exists

- A state E_j to which no return is possible is considered aperiodic

Markov chains

- To deal with a periodic E_j it suffices to consider the chain at the trials $t, 2t, 3t$
- In this way we obtain a new Markov chain with transition probabilities p_{ik}^t
- In this new chain E_j is aperiodic
- Results concerning aperiodic states can thus be transferred to periodic states

Markov chains

Persistent and transient states

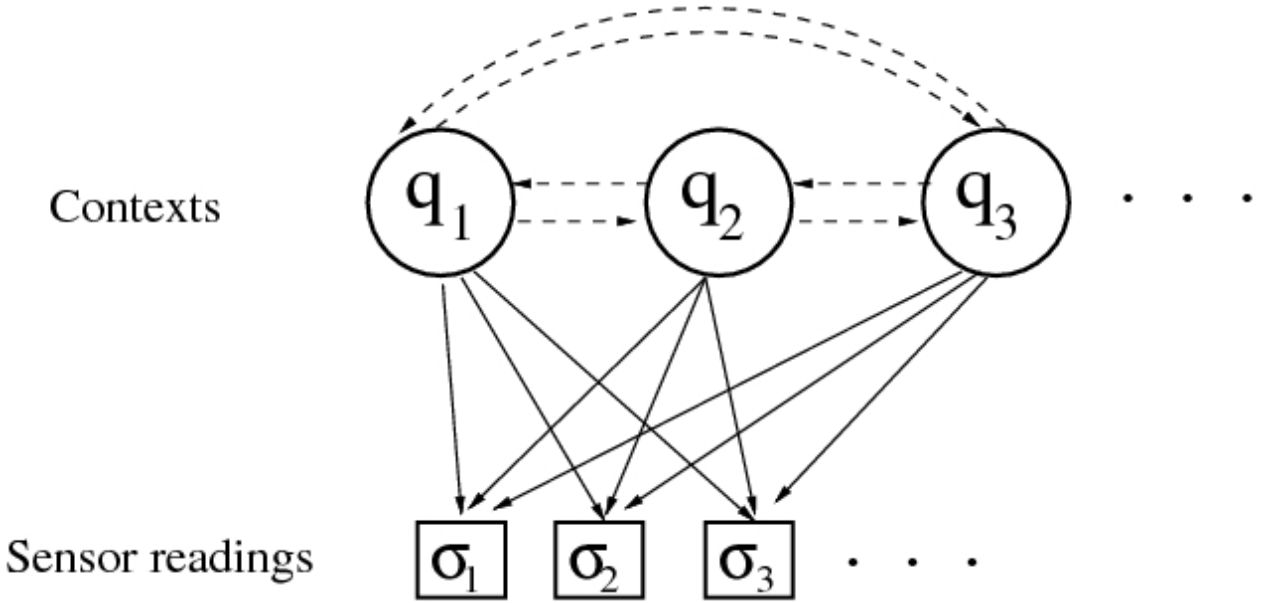
- The state E_j is persistent if $\sum_{n=1}^{\infty} p_{jj}^n = 1$ and transient if $\sum_{n=1}^{\infty} p_{jj}^n < 1$
- A persistent state E_j is called null state if its mean recurrence time $\mu_j = \infty$

Ergodic states

- An aperiodic persistent state E_j with $\mu_j < \infty$ is called ergodic

Hidden Markov Models

- Make a sequence of decisions for a process that is not directly observable[20]
- Current states of the process might be impacted by prior states
- HMM often utilised in speech recognition or gesture recognition



[20] Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern classification*, Wiley interscience, 2001.

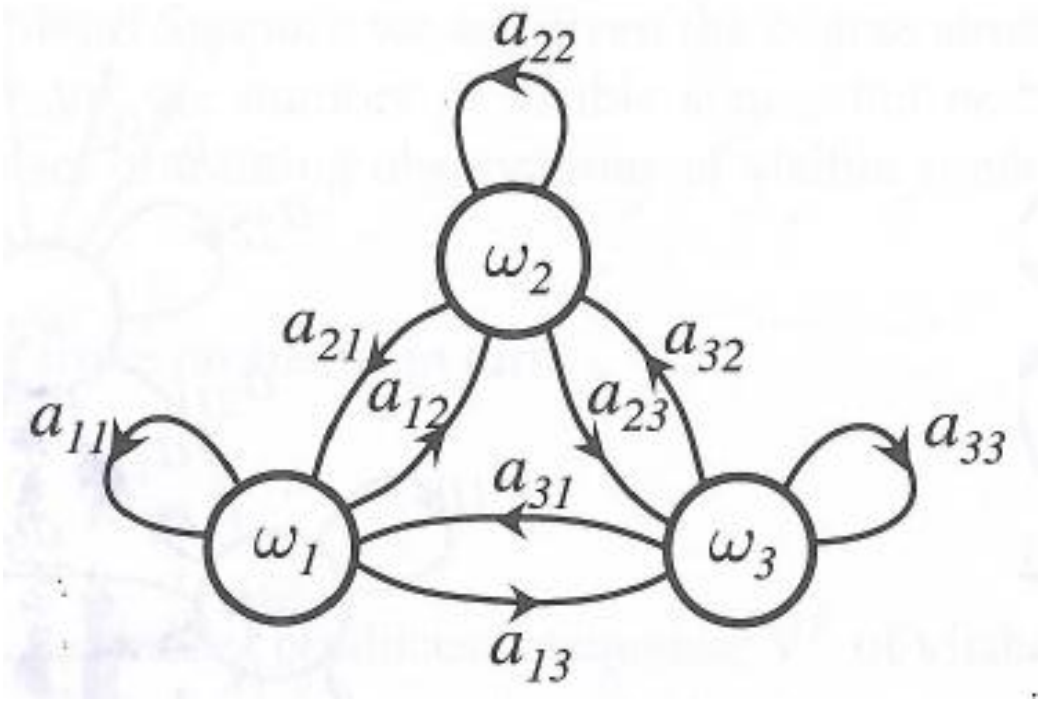
Hidden Markov Models

- Computational biology
  - Align biological sequences
  - Find sequences homologous to a known evolutionary family
  - Analyse RNA secondary structure [21]
- Computational linguistics[22]
  - Topic segmentation of text
  - Information extraction

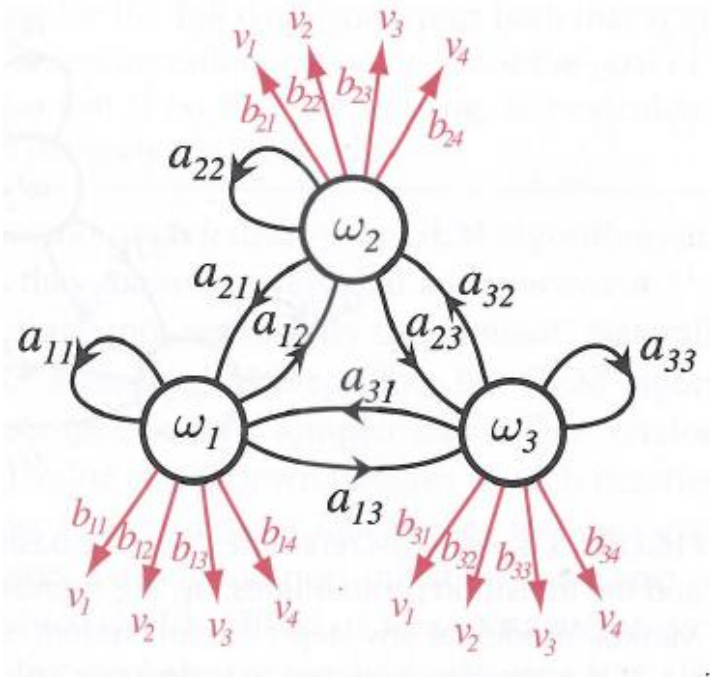
[21] R. Durbin, S. Eddy, A. Krogh and G. Mitchison, *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*, Cambridge University Press, 1998.

[22] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*, MIT Press, 1999.

Hidden Markov Models



Hidden Markov Models



- At every time step t the system is in an internal state $\omega(t)$
- Additionally, we assume that it emits a (visible) symbol $v(t)$
- Only access to visible symbols and not to internal states

Hidden Markov Models

- $V^T = \{v(1), v(2), \ldots, v(T)\}$
- In any state $\omega(t)$ we have a probability of emitting a particular visible symbol $v_k(t)$
- Probability to be in state $\omega_j(t)$ and emit symbol $v_k(t)$:
  - $P(v_k(t)|\omega_j(t)) = b_{jk}$
- Transmission probabilities: $p_{ij} = P(\omega_j(t+1)|\omega_i(t))$
- Emission probability: $b_{jk} = P(v_k(t)|\omega_j(t))$

Hidden Markov Models

- Normalisation conditions

$$\sum_j p_{ij} = 1 \text{for all } i$$

$$\sum_k b_{jk} = 1 \text{for all } j$$

Hidden Markov Models

- Central issues in hidden Markov models:
  - Evaluation problem
    - Determine the probability that a particular sequence of visible symbols V^T was generated by a given hidden Markov model
  - Decoding problem
    - Determine the most likely sequence of hidden states ω^T that led to a specific sequence of observations V^T
  - Learning problem
    - Given a set of training observations of visible symbols, determine the parameters p_{ij} and b_{jk} for a given HMM

Hidden Markov Models – Evaluation problem

- Probability that model produces a sequence V^T:

$$P(V^T) = \sum_{\overline{\omega}^T} P(V^T|\overline{\omega}^T)P(\overline{\omega}^T)$$

- Also:

$$P(\overline{\omega}^T) = \prod_{t=1}^{T} P(\omega(t)|\omega(t-1))$$

$$P(V^T|\overline{\omega}^T) = \prod_{t=1}^{T} P(v(t)|\omega(t))$$

- Together:

$$P(V^T) = \sum_{\overline{\omega}^T} \prod_{t=1}^{T} P(v(t)|\omega(t))P(\omega(t)|\omega(t-1))$$

Hidden Markov Models – Evaluation problem

- Probability that model produces a sequence V^T:

$$P(V^T) = \sum_{\overline{\omega}^T} \prod_{t=1}^{T} P(v(t)|\omega(t))P(\omega(t)|\omega(t-1))$$

- Formally complex but straightforward
- Naive computational complexity
  - $\mathcal{O}(c^T T)$

Hidden Markov Models – Evaluation problem

- Probability that model produces a sequence V^T:

$$P(V^T) = \sum_{\omega^T} \prod_{t=1}^{T} P(v(t)|\omega(t))P(\omega(t)|\omega(t-1))$$

- Computationally less complex algorithm:
  - Calculate $P(V^T)$ recursively
  - $P(v(t)|\omega(t))P(\omega(t)|\omega(t-1))$ involves only $v(t), \omega(t)$ and $\omega(t-1)$

$$\alpha_j(t) = \begin{cases} 0 & t = 0 \text{ and } j \neq \text{ initial state} \\ 1 & t = 0 \text{ and } j = \text{ initial state} \\ \left[\sum_i \alpha_i(t-1)p_{ij}\right] b_{jk} & \text{otherwise } (b_{jk} \text{ leads to observed } v(t)) \end{cases}$$

Hidden Markov Models – Evaluation problem

- Forward Algorithm
- Computational complexity: $O(c^2 T)$

Forward algorithm

```
1 initialise t ← 0, p_ij, b_jk, V^T, α_j(0)
2     for t ← t + 1
3         j ← 0
4         for j ← j + 1
5             α_j(t) ← b_jk Σ_{i=1}^{c} α_i(t − 1) p_ij
6         until j = c
7     until t = T
8 return P(V^T) ← α_j(T) for the final state
9 end
```

Hidden Markov Models – Decoding problem

- Given a sequence V^T, find most probable sequence of hidden states
- Enumeration of every possible path will cost $O(c^T)$
  - Not feasible

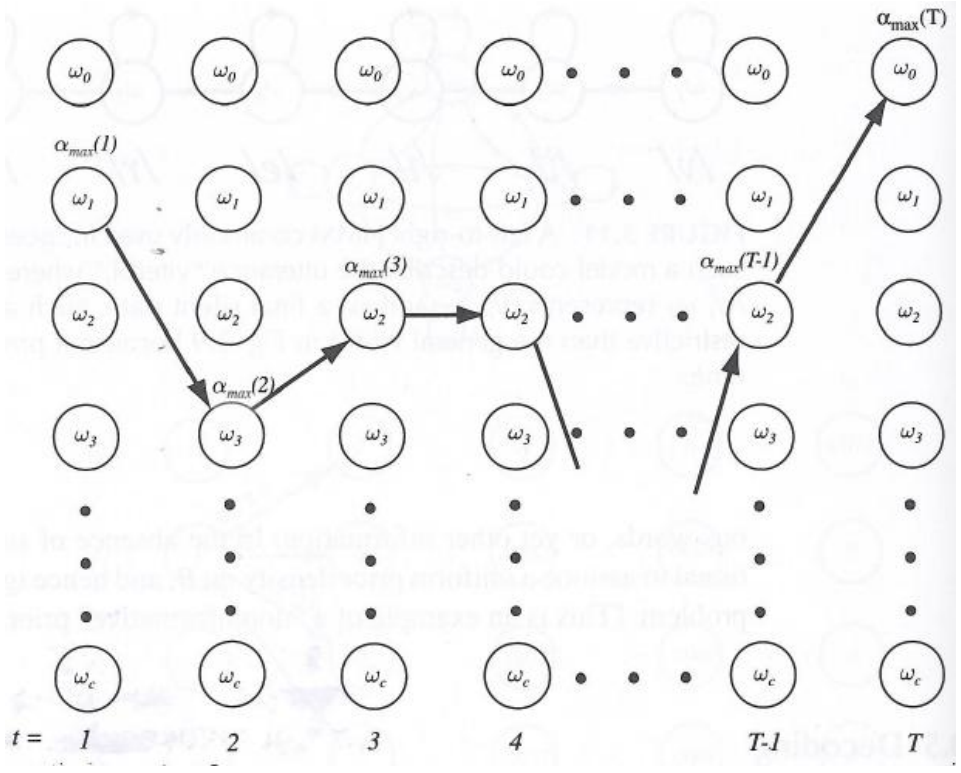
Hidden Markov Models – Decoding problem

- Given a sequence V^T, find most probable sequence of hidden states

Decoding algorithm

```
1  initialise:  path ← {}, t ← 0
2      for t ← t + 1
3          j ← 0;
4          for j ← j + 1
5              α_j(t) ← b_jk Σ_{i=1}^{c} α_i(t − 1) p_ij
6          until j = c
7          j' ← arg max_j α_j(t)
8          append ω_{j'} to path
9      until t = T
10 return path
11 end
```

Hidden Markov Models – Decoding problem



- Computational time of the decoding algorithm
  - $O(c^2 T)$

Hidden Markov Models – Learning problem

- Determine the model parameters p_{ij} and b_{jk}
  - Given: Training sample of observed values V^T
- No method known to obtain the optimal or most likely set of parameters from the data
  - However, we can nearly always determine a good solution by the forward-backward algorithm
  - General expectation maximisation algorithm
  - Iteratively update weights in order to better explain the observed training sequences

Hidden Markov Models – Learning problem

- Probability that the model is in state $\omega_i(t)$ and will generate the remainder of the given target sequence:

$$\beta_i(t) = \begin{cases} 0 & t = T \text{ and } \omega_i(t) \text{ not final hidden state} \\ 1 & t = T \text{ and } \omega_i(t) \text{ final hidden state} \\ \sum_j \beta_j(t+1) p_{ij} b_{jk} & \text{otherwise } (b_{jk} \text{ leads to } v(t+1)) \end{cases}$$

Hidden Markov Models – Learning problem

- $\alpha_i(t)$ and $\beta_i(t)$ only estimates of their true values since transition probabilities p_{ij}, b_{jk} unknown
- Probability of transition between $\omega_i(t-1)$ and $\omega_j(t)$ can be estimated
  - Provided that the model generated the entire training sequence V^T by **any** path

$$\gamma_{ij}(t) = \frac{\alpha(t-1)p_{ij}b_{jk}\beta_j(t)}{P(V^T|\Omega)}$$

- Probability that model generated sequence V^T:

$$P(V^T|\Omega)$$

Hidden Markov Models – Learning problem

- Calculate improved estimate for p_{ij} and b_{jk}

$$\overline{p_{ij}} = \frac{\sum_{t=1}^{T} \gamma_{ij}(t)}{\sum_{t=1}^{T} \sum_{k} \gamma_{ik}(t)}$$

$$\overline{b_{jk}} = \frac{\sum_{t=1, v(t)=v_k}^{T} \sum_{l} \gamma_{jl}(t)}{\sum_{t=1}^{T} \sum_{l} \gamma_{jl}(t)}$$

- Start with rough estimates of p_{ij} and b_{jk}
- Calculate improved estimates
- Repeat until some convergence is reached

Hidden Markov Models – Learning problem

Forward-Backward algorithm

1 initialise p_{ij}, b_{jk}, V^T, convergence criterion $\Delta, t \leftarrow 0$
2 do $t \leftarrow t + 1$
3 compute $\overline{p_{ij}(t)}$
4 compute $\overline{b_{jk}(t)}$
5 $p_{ij}(t) \leftarrow \overline{p_{ij}(t)}$
6 $b_{jk}(t) \leftarrow \overline{b_{jk}(t)}$
7 until $\max_{i,j,k}[p_{ij}(z) - p_{ij}(z-1), b_{jk}(t) - b_{jk}(t-1)] < \Delta$
(convergence achieved)
8 return $p_{ij} \leftarrow p_{ij}(t)$, $b_{jk} \leftarrow b_{jk}(t)$
9 end

Outline

Stochastic methods

- When problem structure is not well known, it might be hard to configure classification methods correctly
- Then, randomised search approaches are an option to find optimum parameters for a classifier
- The search process then virtually becomes a problem of iterative learning
- Search space is then spanned by possible configurations for all distinct parameters

Randomised search approaches

- Most frequently applied in engineering design
  - Assumptions to compute extrema are not fulfilled (e.g. unfriendly/unknown conditions)
  - Difficulties to carry out necessary differentiations
  - Solution to the equations describing all conditions does not always lead to optimum point in the search space
  - Equations to describe conditions are not immediately solvable

Randomised search approaches

- Local random search
- Metropolis algorithm
- Simulated annealing
- Evolutionary algorithms

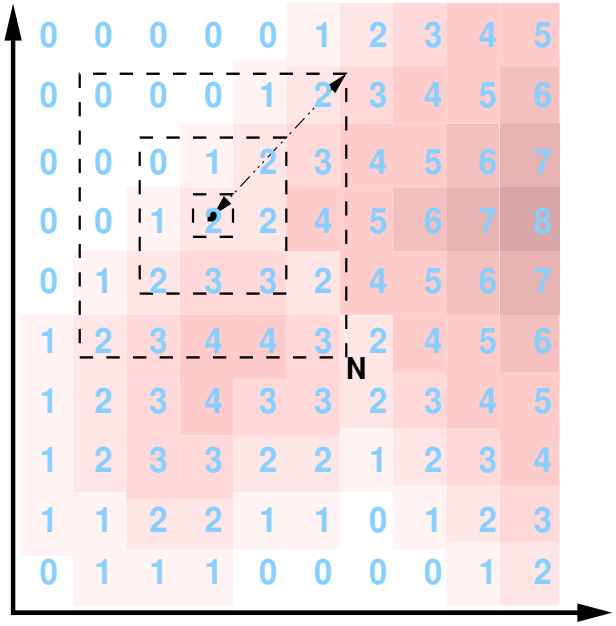
Local random search heuristics

- Local random search
  - Intuitive way to climb a mountain (by a sightless climber)

Local random search

For every point x in a search space S, a non-empty neighbourhood $N(x) \subseteq S$ is defined. The local random search approach iteratively draws one sample $x' \in N(x)$. When the fitness of the new value is better than the old one ($F(x) < F(x')$), the new value is utilised as the new best search point. Otherwise it is discarded.

Local random search heuristics



- In principle, $N(x) = x$ or $N(x) = S$ is valid, but the original idea is that $N(x)$ is a relatively small set of search points.
- The points $x' \in N(x)$ are expected to be nearer to x than those points $x'' \notin N(x)$
- Typically, $x \in N(x)$

Local random search heuristics

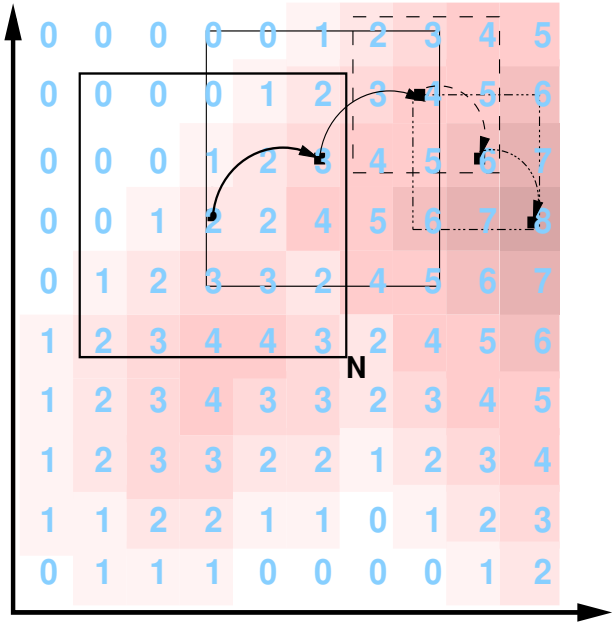
Complexity reduction by restriction of the search space size

- Example: $S = \{0,1\}^n$ and $N_d(x)$ are all points y with Hamming distance smaller than d ($H(x,y) \leq d$)
  - For constant d we obtain: $|N_d(x)| = \Theta(n^d) \ll |S| = 2^n$

$d \leq 1$	$d \leq 2$	$d \leq 3$
1 0 1 0	1 0 1 0	1 0 1 0
0 0 1 0	0 0 1 0	1 1 1 0
1 1 1 0	1 1 1 0	1 1 1 0
1 0 0 0	1 0 0 0	1 0 0 0
1 0 1 1	1 0 1 1	1 0 1 1
	0 1 1 0	0 1 1 0
	0 0 0 0	0 0 0 0
	0 0 1 1	0 0 1 1
	1 1 0 0	1 1 0 0
	1 1 1 1	1 1 1 1
	1 0 0 1	1 0 0 1
		0 1 0 0
		0 1 1 1
		1 1 0 1
		1 0 0 1

$$|N_d(x)| = \binom{n}{d} + \binom{n}{d-1} + \cdots + \binom{n}{1} + \binom{n}{0}$$

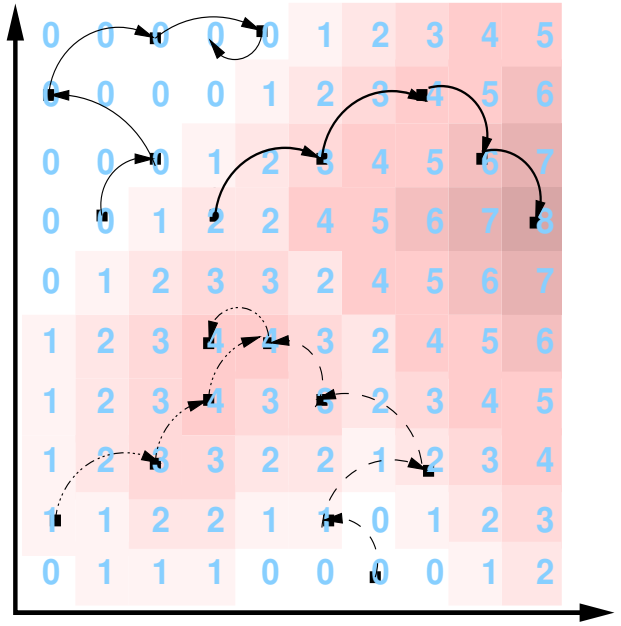
Local random search heuristics



- Small neighbourhood: Easy conversion to local optima
- Huge neighbourhood: Similar to random search
- Change neighbourhood radius during optimisation
  - Initially, big neighbourhood to allow huge steps, then decrease size
  - Challenging: Not to decrease neighbourhood size too fast

Local random search heuristics

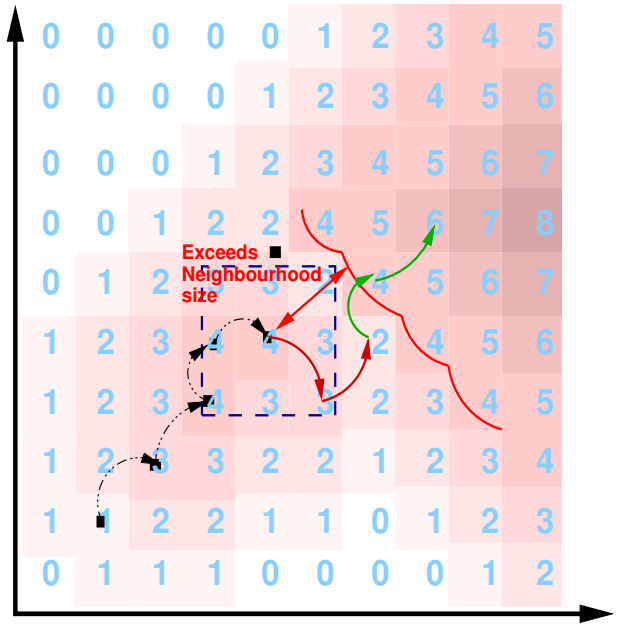
- Alternative to avoid local optima: Multistart strategies
  - Local search approach applied t times on the problem domain
  - Probability amplification results in respectable search result also when single success probability is low.
    - Assume a success probability of $\delta > 0$ for one iteration of the algorithm
    - When the algorithm is applied t times, the overall probability of success is $1 - (1 - \delta)^t$
    - Small polynomial success probabilities are enough for the multistart strategy to obtain very good overall success probabilities



Metropolis algorithms

- For local random search, only multistart strategies can avoid the termination in local optima.
- A Metropolis approach allows to accept also new search points that decrease the fitness value
- If $F(x') < F(x)$ the search point x' is discarded only with probability

$$1 - \frac{1}{e^{(F(x)-F(x'))/T}}$$



Metropolis algorithms

- Probability to accept search points with decreasing fitness value dependent on degree by which fitness decreased
- For $T \to 0$ the Metropolis approach becomes a random search
- For $T \to \infty$ the Metropolis approach becomes an uncontrolled local search
- Choice of T impacts the performance
- Knowledge on the problem or the fitness function might impact the choice of T

Simulated annealing

- Choice of optimal T not easy: Change parameter in the pace of the optimisation
- Initially: T should allow to 'jump' to other regions of the search space with increased fitness value
- Finally: Process should gradually 'freeze' until local search approach propagates the local optimum in the neighbourhood.
- Name chosen in analogy to natural cooling processes in the creation of crystals
  - In this process, the temperature is gradually decreased so that Molecules that could move freely at the beginning are slowly put into their place

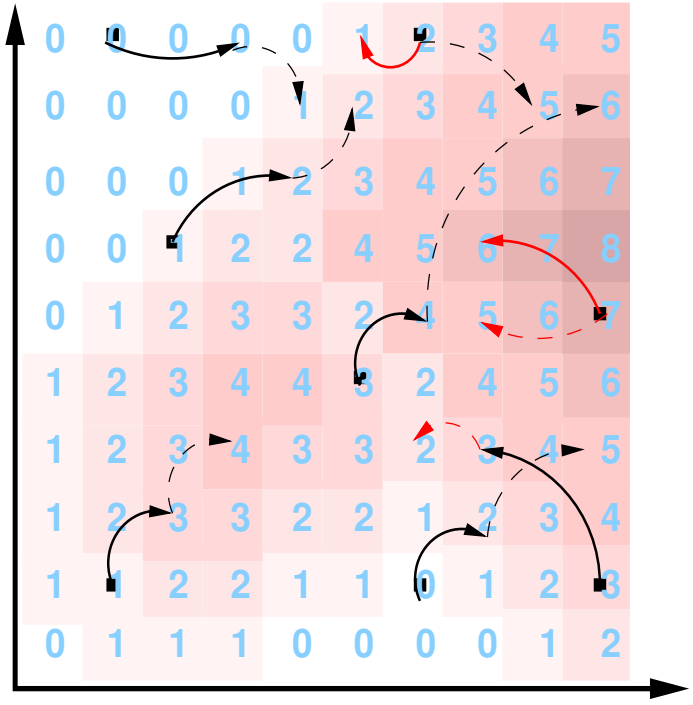
Simulated annealing

- Optimal choice of the cooling schedule for T?
- Non-Adaptive approaches
  - Fixed temperature function $T(t)$
  - Every few steps the original value is multiplied with a factor $\alpha < 1$
- Adaptive approaches
  - React on the optimisation process
  - Probably dependent on the frequency of accepted iterations.

Simulated annealing

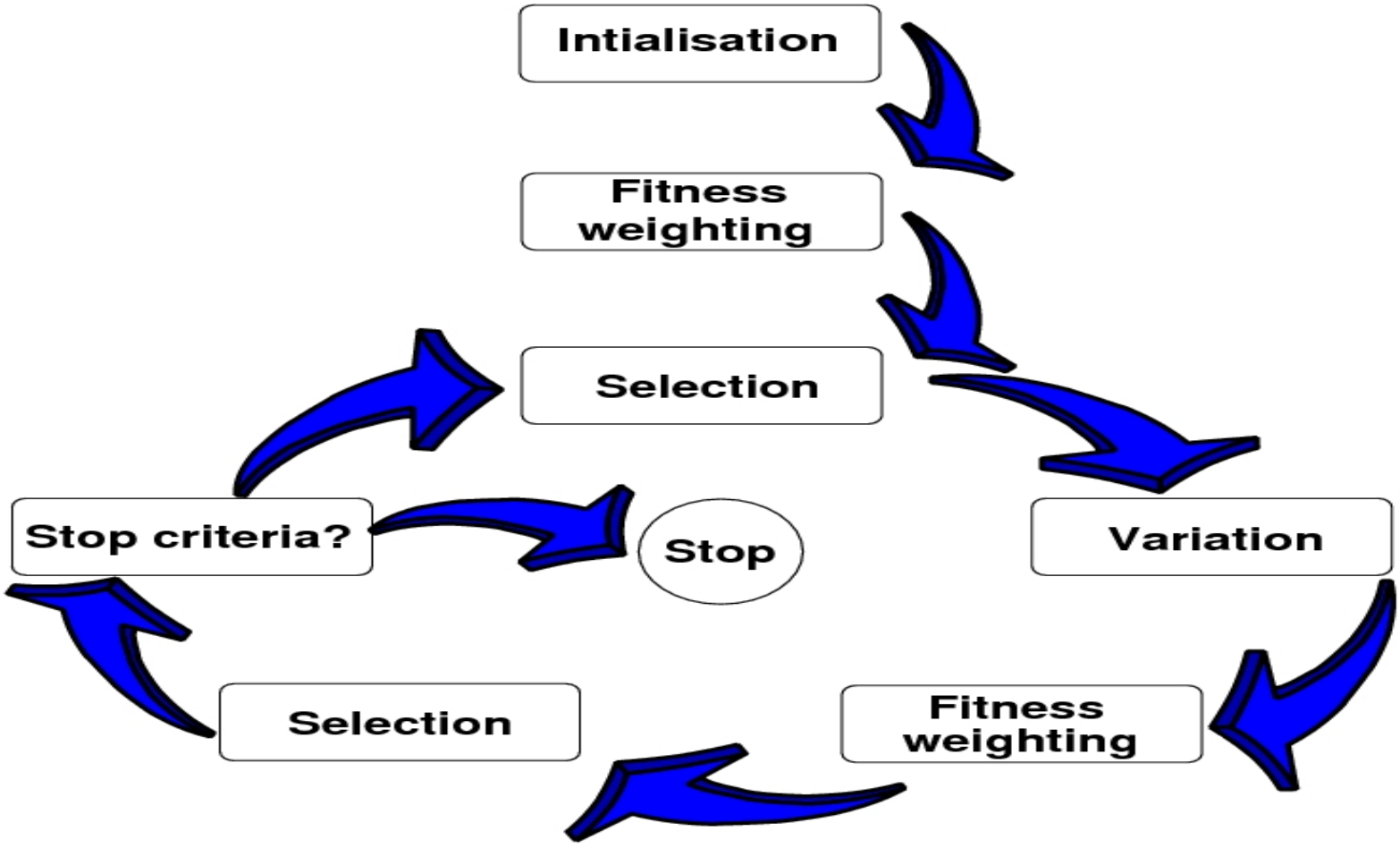
- Problem: No natural problem known for which it has been proved that Simulated Annealing is sufficiently more effective than the Metropolis algorithm with optimum stationary temperature.
- However, artificially constructed problems exist, for which it could be shown that Simulated Annealing is superior to the Metropolis algorithm

Evolutionary algorithms



- Several researchers have studied the use of evolutionary approaches for optimisation purposes
- To-date, evolutionary algorithms combine these different approaches so that no clear distinction can be made

Evolutionary algorithms



Evolutionary algorithms

Genetic algorithms

- Proposed by John Holland [23]
- Binary discrete search spaces: $\{0,1\}^n$
- Fitnessproportional selection
  - For m individuals $x_1, \ldots, x_m$ the probability to choose x_i is $\frac{f(x_i)}{f(x_1)+\cdots+f(x_m)}$.
- Main evolution operator is crossover
  - Originally One-point crossover
- The main goal was not optimisation but the adaptation of an environment

[23] J. Holland: *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

Evolutionary algorithms

Evolution strategies

- Proposed by Bienert, Rechenberg and Schwefel[24] [25]
- At first only steady search spaces as $\mathbb{R}^n$
- No Crossover
- Only mutation
  - First mutation operator: Each component x_i is replaced by $x_i + \sigma Z_i$ (Z_i normally distributed, σ^2 Variance)

[24] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, 1973.*

[25] H.-P. Schwefel, *Evolution and optimum seeking*, 1993

Evolutionary algorithms

Evolutionary programming

- The approach was proposed by Lawrence J. Fogel[26][27]
- Various similarities to evolution strategies
- Search Space: Space of deterministic finite automata that well adapt to their environment.

[26] L.J. Fogel, *Autonomous automata*, Industrial Research, Vol. 4, 1962.

[27] L.J. Fogel, *Biotechnology: Concepts and Applications*, Prentice-Hall, 1963

Evolutionary algorithms

Genetic programming

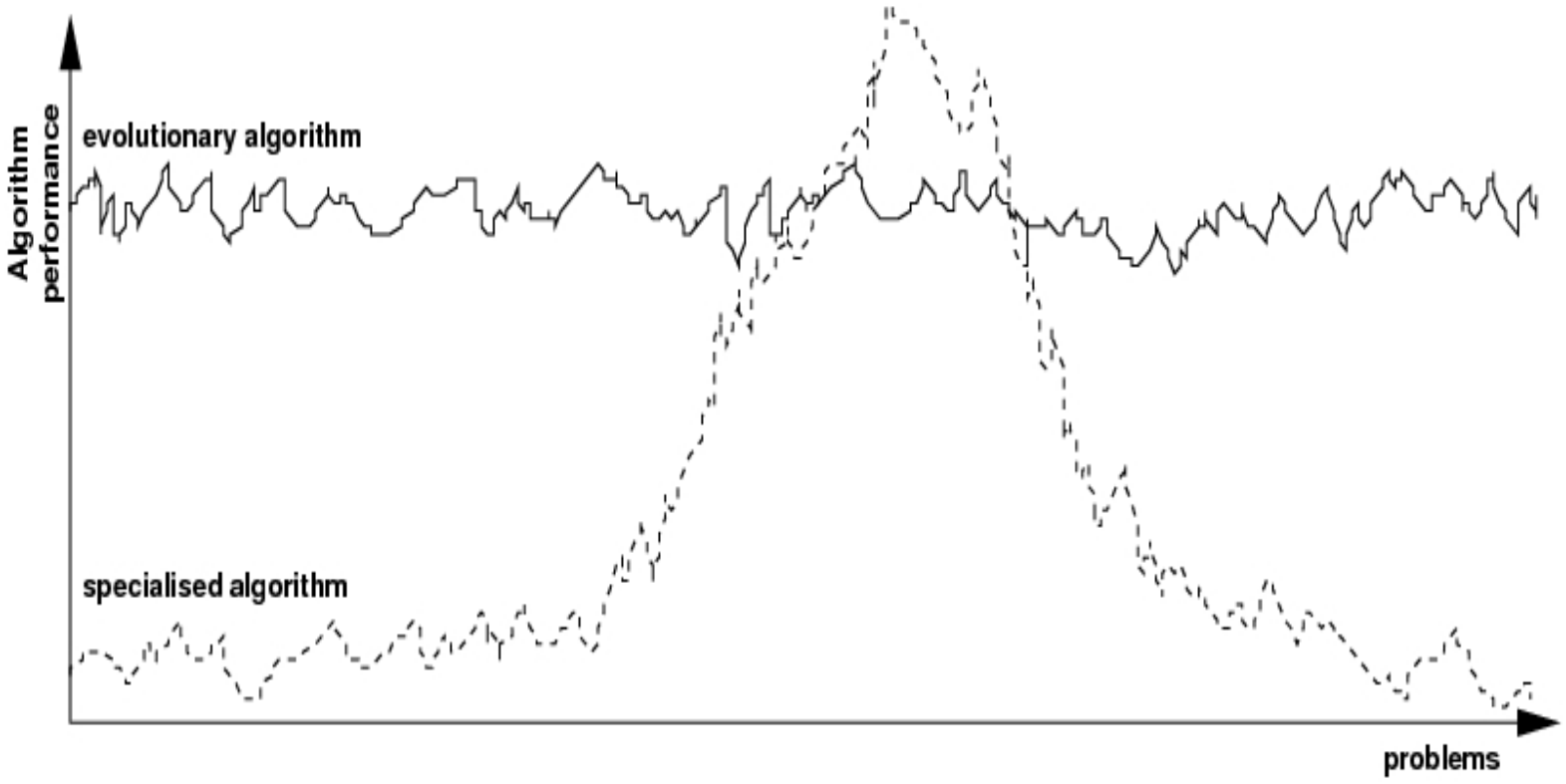
- Proposed by John Koza[28]
- Search space: Syntactically correct programs
- Crossover more important than mutation

[28] John Koza *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992

Restrictions of evolutionary approaches

- In the early days of evolutionary algorithms it has been argued that
  - Problem specific algorithms are better than evolutionary algorithms on a very small subset of problems
  - Evolutionary algorithms perform better on average over all problems
- Therefore, evolutionary algorithms have been proposed as a good choice for a general purpose optimisation scheme

Restrictions of evolutionary approaches



Restrictions of evolutionary approaches

- Can an algorithm be suited for 'all' problems?
  - Distinct coding of the search space
  - Various fitness functions
- What does 'all problems' mean?
  - For all possible representations and sizes of the search space
  - All possible fitness functions on the feature space
  - For a given search space and feature space, all possible fitness functions
  - Every single point in the search space is the optimum point in several of these problems
- Can one algorithm be better on average than another algorithm on 'all' problems?

Restrictions of evolutionary approaches

- To understand this scenario, Wolpert and Macready formalised the assertion[29]
- Assumptions:
  - The set of all functions $f : S \rightarrow W$ considered is given by F
  - S and W are finite (as every computation on physical computers can only have finite resources)
  - The fitness function is evaluated only once for each search point
  - $A(f)$ is the number of search points requested until the optimum is found

[29] D.H. Wolpert and W.G. Macready, *No Free Lunch Theorems for Optimisation*, IEEE Transactions on Evolutionary [illegible] 1997.

Restrictions of evolutionary approaches

No free lunch theorem

Assume that the average performance of an algorithm in the No Free Lunch Scenario for S and W is given by $A_{S,W}$, the average over all $A(f), f \in F$. Given two algorithms A and A', we obtain $A_{S,W} = A'_{S,W}$

- This means that two arbitrary algorithms perform equally well on average on all problems

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

W.l.o.g.: $W = \{1, \ldots, N\}$

We consider sets $F_{s,i,N}$ of all functions f on a search space of non-visited search points of size s with at least one x with $f(x) > i$

Observe that for every function f and every permutation π also f_π belongs to $F_{s,i,N}$

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Proof by induction over $s := |S|$.

Induction start: $s = 1$

Every algorithm has to choose the single optimum search point with its first request.

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$

We define a function $a : S \rightarrow \mathbb{N}$ so that for every $x \in S$ the share of functions with $f(x) = j$ is exactly $a(j)$.

This is independent of x, since all permutations f_π of a function f also belong to $F_{s,i,N}$,

$a(j)$ is therefore the probability to choose a search point with fitness value j – Independent of the concrete algorithm A

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

Count of functions $f(x) = j$ is equal to the number of functions $f_\pi(y) = j$, since all permutations of f are also in $F_{s,i,N}$.

The probability to achieve a fitness value $j > i$ is therefore independent of the algorithm.

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

If $j \leq i$, x is not optimal in scenario $F_{s,i,N}$ and the new scenario is $F_{s-1,i,N}$

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Summary – in other words:

For any two algorithms we can state a suitable permutation of the Problem-function for one problem (i.e. state another problem), so that both algorithms in each iteration request identical search points.

- Especially, since every search point could be optimal, there are always algorithms that request the optimal search point right from the start.

Restrictions of evolutionary approaches

- The NFL is possible, since ALL algorithms and ALL problems are considered
- It is a reasonable question if an NFL is also valid in smaller, more realistic scenarios.
- In [30] is was proved, that a similar theorem can be stated also for more realistic problem scenarios.

[30] S. Droste, T. Jansen and I. Wegener, *Perhaps not a free lunch but at least a free appetizer*, Proceedings of the 1st ...onary Computation Conference, 1999.

Design aspects of evolutionary algorithms

Selection principles

Uniform selection

Individuals chosen uniformly at random

Deterministic selection

Deterministically choose the highest rated individuals for the selection

Threshold selection

Candidates for offspring population drawn uniformly at random from the t highest rated individuals

Design aspects of evolutionary algorithms

Selection principles

Fitnessproportional selection

- For population $x_i, \ldots, x_n$ individual x_i chosen with

$$p(x_i) = \frac{f(x_i)}{f(x_1) + \cdots + f(x_n)}$$

- Draw random variable u from $[0, 1]$ and consider x_i if

$$p(x_1) + \cdots + p(x_{i-1}) < u \leq p(x_1) + \cdots + p(x_i)$$

- Frequently applied for evolutionary approaches

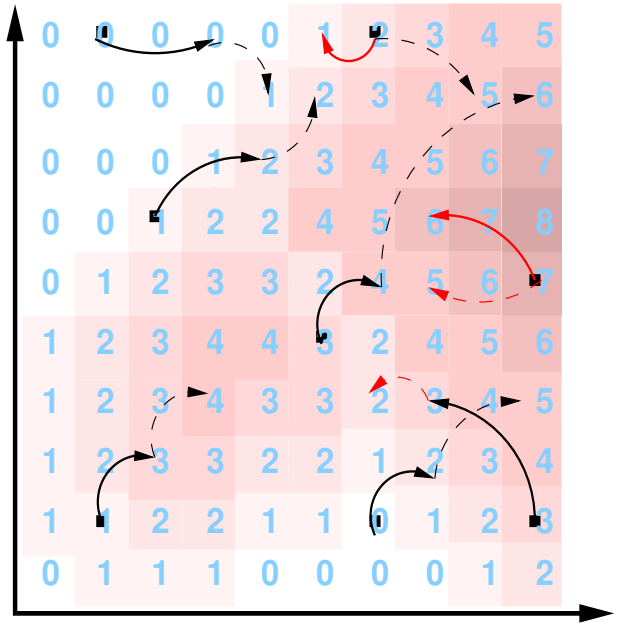
Design aspects of evolutionary algorithms

Selection principles

- Problems with Fitnessproportional selection
  - Linear modification of the fitness function ($f \rightarrow f + c$) results in different behaviour
  - When fitness values sufficiently separated, selection is nearly deterministic
  - When deviation in fitness values is small relative to absolute values, similar to uniform selection

Design aspects of evolutionary algorithms

Variation – Mutation



- Mutation creates one offspring individual from one individual
- Operators are designed for specific search spaces
- Shall apply only few modifications of individuals on average
- Distant individuals have smaller probability

Evolutionary algorithms

Mutation operators for individuals from $\mathbb{B}^n$ (similar operators for other search spaces):

Standard bit mutation

- Offspring individual created bit-wise from parent individual
- Every bit 'flipped' with probability p_m
- Common choice: $p_m = \frac{1}{n}$

1 bit mutation

- Offspring individual identical in all but one bit.
- This bit chosen uniformly at random from all n bits

Design aspects of evolutionary algorithms

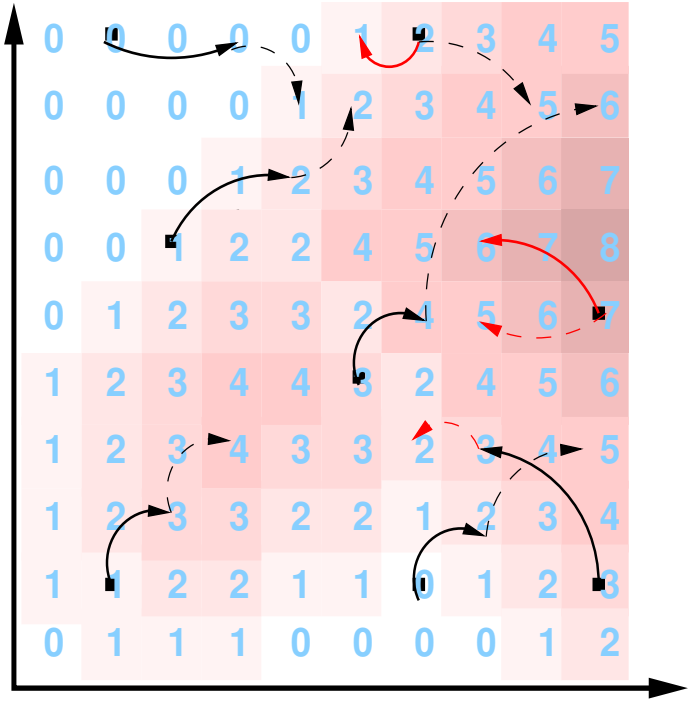
Variation – Crossover

- Crossover typically takes two individuals and results in one or two offspring individuals
  - Also crossover of more than two individuals possible
  - Often generalisations of the two-individual case
- Distinct crossover methods for various search spaces
- Crossover parameter p_c specifies the probability with which crossover (and not mutation) is applied for one selected individual
- In some cases (e.g. binary coded numbers) not all positions in the individual string are allowed to apply crossover on

Design aspects of evolutionary algorithms

Typical crossover variants

- One-point crossover
- k-point crossover
- Uniform crossover



Evolutionary algorithms

Crossover operators for $\mathbb{B}^n$

(Operators for other search spaced similar)

One-point crossover: Individual x'' from two individuals x and x' according to uniformly determined crossover position:

$$x''_j = \begin{cases} x_j & \text{if } j \leq i \\ x'_j & \text{if } j > i \end{cases}$$

Evolutionary algorithms

Crossover operators for $\mathbb{B}^n$

k-point crossover: Choose $k \leq n$ positions uniformly at random:

$$
\begin{array}{rcl}
x_1 = & x_{11}, x_{1,2}, \ldots, x_{1,k_1} \,|\, x_{1k_1+1}, \ldots, x_{1k_2} \,|\, x_{1k_2+1}, \ldots, x_{1n} \\
x_2 = & x_{21}, x_{2,2}, \ldots, x_{2,k_1} \,|\, x_{2k_1+1}, \ldots, x_{2k_2} \,|\, x_{2k_2+1}, \ldots, x_{2n} \\
\hline
y_1 = & x_{11}, x_{1,2}, \ldots, x_{1,k_1} \,|\, x_{2k_1+1}, \ldots, x_{2k_2} \,|\, x_{1k_2+1}, \ldots, x_{1n} \\
y_2 = & x_{21}, x_{2,2}, \ldots, x_{2,k_1} \,|\, x_{1k_1+1}, \ldots, x_{1k_2} \,|\, x_{2k_2+1}, \ldots, x_{2n}
\end{array}
$$

Evolutionary algorithms

Crossover operators for $\mathbb{B}^n$

Uniform crossover: Each bit chosen with uniform probability from one of the parent individuals

Design aspects of evolutionary algorithms

Discussion

- Evolutionary algorithms are easy to implement when compared to some complex specialised approaches
- However, Evolutionary algorithms are computationally complex
- It is therefore beneficial to implement efficient variants to the distinct methods

Design aspects of evolutionary algorithms

Discussion

- Generation of pseudo random bits is important for many of the theoretic results for evolutionary algorithms to hold
- It is, however possible to reduce the number of random experiments
  - It is more efficient to calculate the next flipping bit in a mutation instead of doing the calculation for every bit independently

Design aspects of evolutionary algorithms

Discussion

- Most of the computational time is typically consumed by the fitness calculation
- One approach to reduce complexity is to prevent re-calculation of fitness for individuals
  - Dynamic data structures that support search and insert

Outline

Questions?

Stephan Sigg
sigg@ibr.cs.tu-bs.de

Literature

- C.M. Bishop: Pattern recognition and machine learning, Springer, 2007.
- P. Tulys, B. Skoric, T. Kevenaar: Security with Noisy Data – On private biometrics, secure key storage and anti-counterfeiting, Springer, 2007.
- R.O. Duda, P.E. Hart, D.G. Stork: Pattern Classification, Wiley, 2001.